PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

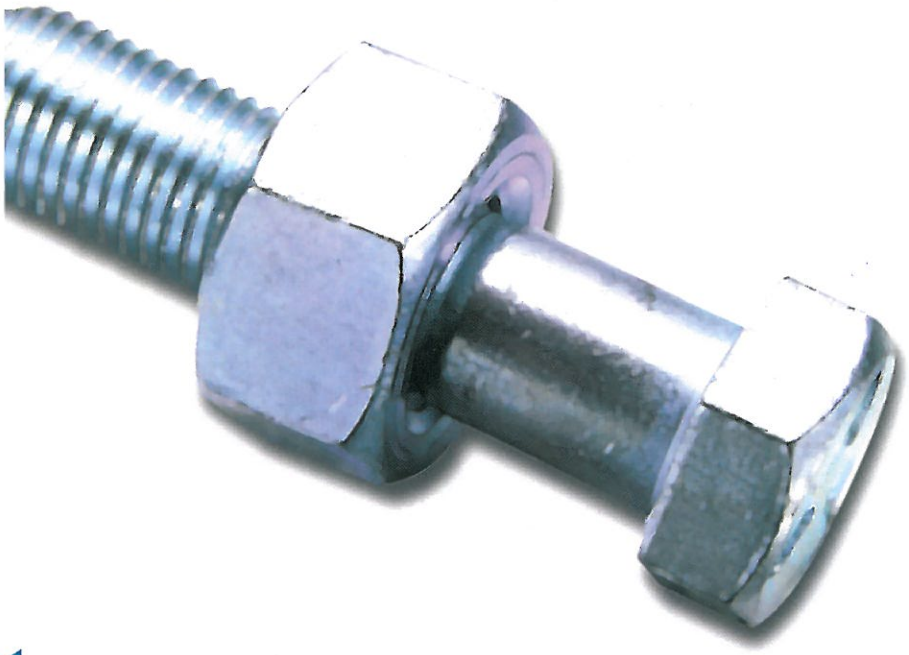
SOLO PROGRAMADORES

IV, NÚMERO 44

975 Ptas.

CONEXIÓN CON INTERNET

PROGRAMACIÓN DE SOCKETS



CONTENIDO DEL CD-ROM

- Personal Oracle 8
- Instant Basic 4 JavaTM
- SmallTalk
- HTML 4.0 Reference Library
- Modula-3
- i.Share 2.5

JAVA
Servlets: La cara oculta de la Web

LINUX
Configuración de clientes X con Motif Windows Manager

SISTEMAS DISTRIBUIDOS
La librería de acceso

DIRECTX
Programación con Delphi

INTERNET CON LINUX
Servicio de correo electrónico

WINDOWS NT SERVER
Internet Information Server: Configuración

PROGRAMACIÓN HARDWARE
Control del módem con Visual Basic

VISUAL BASIC
Los controles ActiveX

8 413042 303299

SÓLO PROGRAMADORES

Número 44
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo de Riquer
Redactora Jefe
Lucia R. Muñoz
lucirm@towercom.es
Colaboradores
Enrique Díaz, José Antonio Mendoza, Ernesto Schmitz, Chema Alvarez, Carlos Alvaro, Constantino Sánchez Ballesteros, Arturo Alvaro, Jorge F. Delgado, Xesco Hernández,Juan Carlos Sastre, Enrique da la Lastra, Jordi Agost
Maquetación
Susana Llano
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Begoña Alvarez
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Óscar Rodríguez (jefe)
oscarrf@towercom.es
Javier Amado
jamado@towercom.es
Servicio Técnico
Óscar Casado
ocasado@towercom.es

Filmación
Megatipo
Impresión
Gráficas Reunidas
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 31-06-98

EDITORIAL

Ser o no ser, o al menos parecerlo

Dentro del vertiginoso mundo de la Informática, y concretamente en el de la programación, las nuevas tecnologías han conseguido marear a la mayoría de sus incondicionales. Además del alto grado de especialización existente en cada una de las numerosas áreas, parece que los fabricantes se ponen de acuerdo (o más bien al contrario) para que nuestro trabajo se convierta en una difícil elección. En lugar de plantear alternativas que converjan en una única tecnología o propuesta estandarizada, nos obligan a caminar por el espinoso camino de la dualidad: UNIX/NT, Java sí o no, servlets/CGI, CORBA/DCOM, etc. Una vez que las partes admiten la idoneidad de cierto concepto, surge la guerra por la imposición del estándar particular, desembocando en la aparición de soluciones propietarias, e incompatibles. Sea cual sea la naturaleza del proyecto, la cantidad de encrucijadas es de tal calibre que los sufridos programadores no tenemos tiempo de conocer, e implantar cierta técnica novedosa.

Ante esta realidad tenemos dos alternativas (además del llanto): alcanzar un alto grado de especialización o realizar un aprendizaje continuo en relación con los temas más candentes. Ambas posturas tienen sus detractores, puesto que alcanzar el estatus de experto conlleva un alejamiento de los últimos avances. El segundo camino, requiere un esfuerzo enorme y nos puede hacer sentir como aprendices de todo y maestros de nada.

Posiblemente la mejor solución pasa por navegar entre dos aguas, manteniendo el reciclaje continuo informados en todo momento de los adelantos del sector, pero enfocando nuestros esfuerzos en un conjunto restringido de temas con un futuro prometedor. Esta visión, junto con la preparación personal al margen de universidades (con planes de estudio incapaces de asimilar un ritmo tan trepidante), se convertirá en una de las mejores virtudes de un profesional cualificado y con plena vigencia. Por este motivo, Sólo Programadores intenta combinar todos los temas existentes dentro del mundo de la programación, alternando conceptos de última aparición con tecnologías totalmente asentadas. En algunos casos se hace más hincapié en el concepto descrito que en la herramienta de desarrollo utilizada. En otros, el lenguaje o herramienta se convierte en el propio fin del artículo, debido a su relevancia o demanda dentro del sector empresarial. En cualquier caso, nuestro objetivo pasa por asentar los conceptos teóricos con la inestimable ayuda de las explicaciones prácticas, con el fin de no perder el ritmo marcado por una actualidad frenética.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: solop@towercom.es, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos. Sois nuestros mejores colaboradores.

solo PROGRAMADORES

44

6 Noticias y Libros
NOVEDADES DEL MERCADO

Este mes podremos encontrar en esta sección la información relacionada con la nueva actualización de Brtieve Pro, el nuevo módulo de procesador UltraSPARC II de Sun...y las últimas publicaciones de programación, más actuales del mercado.

11 DirectX
PROGRAMACIÓN DE DIRECTX CON DELPHI (IV)

En este artículo perteneciente a la serie DirectX el lector tendrá ocasión de conocer los últimos modos Hi-Color y Windowed mode, implementados a la versión DGC para Delphi 3.0. Permiten incorporar numerosos colores a los gráficos, y la ejecución de los juegos en una ventana de Windows.

19 Visual Basic
LOS CONTROLES ACTIVEX (I)

Se aborda la creación de controles detalladament junto al tiempo de vida del mismo y las propiedades c las que dispone. Con ello se ofrece una visión glob sobre este tema habiendo precedido a éste artícu en números anteriores la manera en que se agrega clases al control ActiveX.

25 Programación de Sockets
CREACION DE UN CLIENTE ECHO

Con el manejo de estos elementos, se pretende programar una aplicación completa en Internet. En este artículo se explicará un ejemplo de utilización de uno de estos componentes, desarrollados en un cliente echo simple.

38 Java
SERVLETS (I): LA CARA OCULTA DE LA WEB

Analizando los servlets se da comienzo a esta nueva serie de Java. Gracias a ella conoceremos la "cara oculta", de este lenguaje de programación, tratando algunas cuestiones tales como la sustitución de las CGI´s, la configuración de las variables de su entorno y aquellas aplicaciones que se podrían implementar mediante los servelts.

41
Sistemas distribuidos
LA LIBRERIA DE ACCESO

Con el tratamiento de los sistemas distribuidos se hará frente a los problemas que plantea para el programador utilizar varias máquinas para ejecutar una determinada aplicación. Mediante este análisis se dará respuesta a estas dificultades.

49
Linux
CONFIGURACIÓN DE CLIENTES X CON MOTIF WINDOWS MANAGER

Analizamos este tipo de clientes que son cada vez más habituales en la creación de aplicaciones y más solicitadas en cualquier tipo de entornos de trabajo. En este artículo se verán una serie de técnicas orientadas a su configuración y sus aplicaciones.

54
Internet con Linux
SERVICIO DE CORREO ELECTRÓNICO

Profundizamos en el funcionamiento del servicio de transferencia de correo electrónico, tratando el cliente, servidor y protocolo, la utilización detallada de SMTP, y todos los aspectos relacionados con el uso y configuración del cliente-mail.

59
Programación Hardware
CONTROL DEL MÓDEM CON VISUAL BASIC

Con el análisis del control MSComm, se proporciona un estándar de comandos de comunicaciones. Se van a explicar entre otros aspectos, en este artículo los fundamentos de este tipo de comunicación, la manera de establecer una conexión y el trabajo realizado con un módem.

68
Windows NT Server
INTERNET INFORMATION SERVER (I): CONFIGURACIÓN BÁSICA

En este primer capítulo de la serie IIS, se explicará la aplicación basada en Windows NT. Mediante la utilización de este nuevo servidor se dará servicio WWW, FTP y Gopher, de un modo simultáneo.

74 Programación científica
INTEGRACIÓN NUMÉRICA

Profundizando en este tema se mostrará en el artículo dos visiones matemáticas diferentes: la cuadratura adaptativa que está basada en Simpson, como basada en la ecuación de Gauss-Legendre, realizando una comparación entre ambos métodos.

80
CORREO DEL LECTOR

En esta sección se pretende dar respuesta a todas las dudas que puedan tener nuestros lectores, pudiendo dirigirse a los propios autores o bien a la dirección de correo de nuestra publicación: solop@towercom.es.

81
CONTENIDO DEL CD-ROM

En esta ocasión el CD incluye el entorno de programación Modula-3, el lenguaje de programación orientado a objetos, Smalltalk...Unido a la última revisión de la especificación definitiva de HTML, y otros aspectos innovadores de la programación, acompañados de las fuentes de algunos de los artículos de esta publicación.

Noticias

Btrieve Pro 7

MEJORA EN EL ACCESO A DATOS

La compañía desarrolladora de software, Pervasive Software Inc, ha presentado recientemente Btrieve Pro 7.

Btrieve Pro 7 es la nueva actualización de la solución de base de datos embebible que ofrece la mejor solución en lo que se refiere al acceso a datos tanto del mundo transaccional como relacional desde un motor compartido.

Esta arquitectura permite a las aplicaciones cliente/servidor empaquetadas, acceder a un único motor de bases de datos con el rápido proceso transaccional de alto volumen y las potentes capacidades SQL estándares, de forma simultánea.

Btrieve Pro 7 entrega una arquitectura fundamental de los productos Btrieve y Scalable SQL de Pervasive.

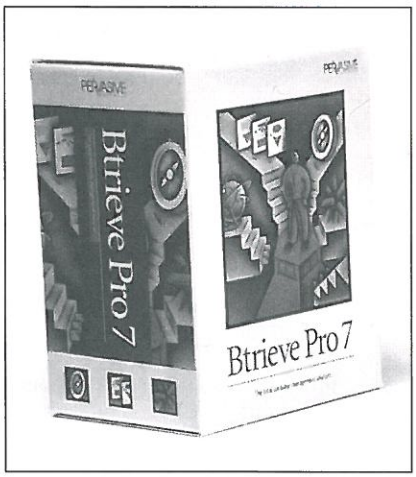
Incluye esta arquitectura, por otro lado, una amplia gama de nuevas características para de este modo lograr una mejora notable de las prestaciones, una fácil instalación y mantenimiento, además de conseguir menor coste de propiedad y compatibilidad con la utilización de las aplicaciones existentes que están basadas en Pervasive.

Las prestaciones de Btrieve Pro 7 se ven incrementadas hasta diez veces más en lo que respecta a los anteriores productos Pervasive. Esta mejora se produce gracias a la nueva tecnología de transporte y administración de hilo microkernel. Estas transformaciones se combinan con la optimización específica de la plataforma.

Al mismo tiempo un motor de bases de datos mucho más rápido acelera las prestaciones de las aplicaciones,permitiendo de esta manera un acceso más eficaz para el usuario y mejorando la escalabilidad a un mayor número de cuentas pertenecientes a los usuarios.

Las nuevas utilidades de instalación y mantenimiento, que se pueden encontrar en el mercado, tales como InstallScout, convierten a Btrieve Pro 7 en una verdadera solución cliente/servidor plug and play.

Esta característica guía a los usuarios finales ayudándoles en aquellos problemas típicosy cotidianos. Para rebajar todavía más los costes de propiedad , las utilidades de configuración y motorización, pueden éstos ser utilizados a distancia.



Como característica destacable de Btrieve Pro 7 se puede mencionar que es totalmente compatible con anteriores versiones de Btrieve y Scalable SQL.

Esta herramienta está a disposición del usuario y programador, a nivel mundial mediante el canal de distribución de Pervasive en plataformas Microsoft Windows NT, Novell NetWare con soporte cliente para DOS, Windows NT, 95 y 3.x.

Finalmente el precio de la licencia para diez usuarios es de 145.000 pesetas aproximadamente.

Sun Microsystems
NUEVO MÓDULO DEL PROCESADOR ULTRASPARC II

Sun Microsystems ha anunciado un nuevo módulo del procesador UltraSPARC II a 336 MHz., con una memoria externa de 4 Mb.. Este módulo se encuentra disponible para la línea actual de servidores de gama media y gama alta, los servidores Sun Enterprise, modelos 3000 al 10000 así como para los Sun Enterprise Cluster.

Con la presentación de este nuevo módulo, Sun ofrece una vez más a sus clientes una vía de actualización clara que les permite mejorar sus sistemas actuales, a la vez que pueden mantener la inversión.

Con el incremento de la velocidad del procesador se mejora el rendimiento global del sistema, así como los tiempos de respuesta de aquellos usuarios que consideren que sus sistemas están alcanzando los niveles máximos de rendimiento. Además los nuevos módulos de CPU, más rápidos, ofrecen un mayor rendimiento a los servidores existentes, lo que permite a los departamentos de sistemas de información aumentar el número de usuarios a medida que crezca el negocio. Con la presentación de este nuevo módulo de procesador a 336 MHz, Sun también ha reforzado la continuidad de su familia en cuanto a procesadores UltraSPARC.

La familia Enterprise de Sun de servidores de la gama media y alta es escalable desde seis hasta 64 procesadores, y ofrece a los clientes valor y fiabilidad excepcionales para aplicaciones críticas.

Todos los servidores Enterprise soportan el entorno operativo Solaris, que ofrece interoperabilidad multifabricante y una escalabilidad total en la línea de productos.

Si desea más información sobre estos productos de Sun Microsystems puede acceder a la página Web **www.sun.es**

Internet Explorer 4.0 para Sun Solaris 2.5
NUEVAS PRESTACIONES PARA EL SISTEMA UNIX

Microsoft anuncia la disponibilidad a nivel mundial de Microsoft Internet Explorer 4.0 para Sun Solaris 2.5 y nuevas versiones de la plataforma UNIX de Solaris.

Al estar construido específicamente para la plataforma UNIX, el navegador consigue unas prestaciones fácilmente reconocibles por los usuarios de este S.O.

Por ejemplo, el interfaz de usuario proporciona la posibilidad a los usuarios de adaptarse rápidamente a Internet Explorer sin tener que volver a aprender las funciones más simples.

Se trata de una aplicación extensible, de forma que los usuarios pueden conectarse con sus aplicaciones Internet favoritas para correo electrónico, noticias o edición, llamándolas directamente desde su navegador.

Esta nueva versión de Explorer, proporciona a los usuarios de Solaris un conjunto de características únicas, incluyendo las innovadoras Barras de Exploración que simplifican la tarea de buscar aquello que se necesita de la Web, las Zonas de Seguridad, soporte HTML Dinámico basado en estándares.

Por otro lado ofrece funciones de accesibilidad que permiten la plena utilización de Internet Explorer interactuando solamente con el teclado.

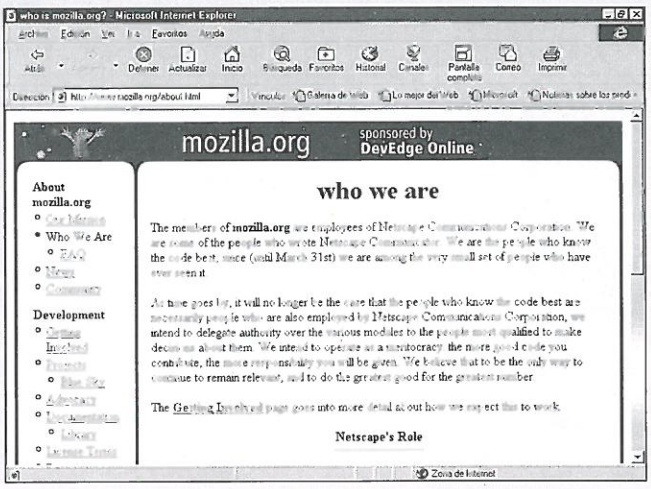
La versión de Explorer para UNIX, incluye nuevas mejoras, como por ejemplo el AutoCompletado, permitiendo a los usuarios teclear las direcciones Web más rápidamente. Aprovechando las barras de exploración, los usuarios pueden localizar, gestionar y navegar fácilmente por sus sitios Web favoritos. También incluye un Soporte de Contenidos de Canal Activo.

Este lanzamiento de la versión definitiva de Internet Explorer 4.0 para UNIX permite a las empresas evaluar y desplegar toda la familia de navegadores en sus empresas.

Internet Explorer 4.0 para Solaris está disponible de forma gratuita en **www.microsoft.com/ie/ie40/.** Si desea ampliar la información sobre el navegador puede consultar en la página **www.microsoft.com/ie/.**

Mozilla.org
UNA WEB SÓLO PARA PROGRAMADORES

Netscape ha anunciado la creación de un Web, mozilla.org, que promocionará, fomentará y guiará a un diálogo abierto y al desarrollo del código fuente cliente de Netscape. Netscape será el primer desarrollador que libere gratis el código fuente de Communicator 5.0, y mozilla.org actuará como un punto focal para desarrolladores que estén interesados en modificar y redistribuir el código fuente cliente de Netscape en sus aplicaciones.



El escenario web proveerá un punto central de contacto y encuentro para animar a los desarrolladores a descargar el código fuente cliente, enviar sus mejoras, tomar parte en grupos de discusión y obtener y compartir información sobre Communicator con Netscape y otros colectivos.

Con esta liberalización del código fuente, Netscape pretende extender su liderazgo en software de cliente Web e integrar las mejoras que puedan aportar los desarrolladores. Para la compañía el éxito y popularidad de Apache, el sistema operativo Linux o la versión BSD de UNIX, prueban el valor y el impacto del desarrollo de código fuente abierto.

Esta iniciativa comenzará a finales del primer trimestre de 1998, coincidiendo con el lanzamiento del primer desarrollo del producto.

La página Web de mozilla.org es también accesible mediante el site de desarrolladores de Netscape en la siguiente dirección: **developer.netscape.com**

Sun e IBM
CERRADO EL ACUERDO PARA EL DISEÑO DE CHIPS JAVA

Sun e IBM empresas líderes en el sector informático llegaron a un acuerdo a principios del pasado mes de marzo.

Gracias a este acuerdo comercial entre ambas compañías se fabricarían de forma conjunta una serie de chips.

Estos chips podrán ser utilizados para teléfonos móviles o bien para smart cards.

Según las últimas informaciones referidas a este tema Sun también autorizará su diseño a empresas como LG, Mitsubishi, NEC y Samsung.

Estos procesadores, construidos bajo el núcleo la arquitectura picoJava, podrán ejecutar todas las instrucciones de una máquina virtual Java.

Además no necesitan disponer de un compilador en tiempo real de este lenguaje.

Según fuentes de Sun "estamos autorizando su arquitectura para crear mercado para estos chips".

Intel y Microsoft
PROYECTO PARA ESTANDARIZAR LAS WORKSTATION WINTEL

Intel y Microsoft han diseñado una nueva estrategia en cuanto a estaciones de trabajo.

El nuevo proyecto planteado por ambas compañías ayudará a los usuarios y desarrolladores a migrar fácilmente sus estaciones de trabajo y el software RISC/UNIX a plataformas con arquitectura Intel y sistema operativo Windows NT.

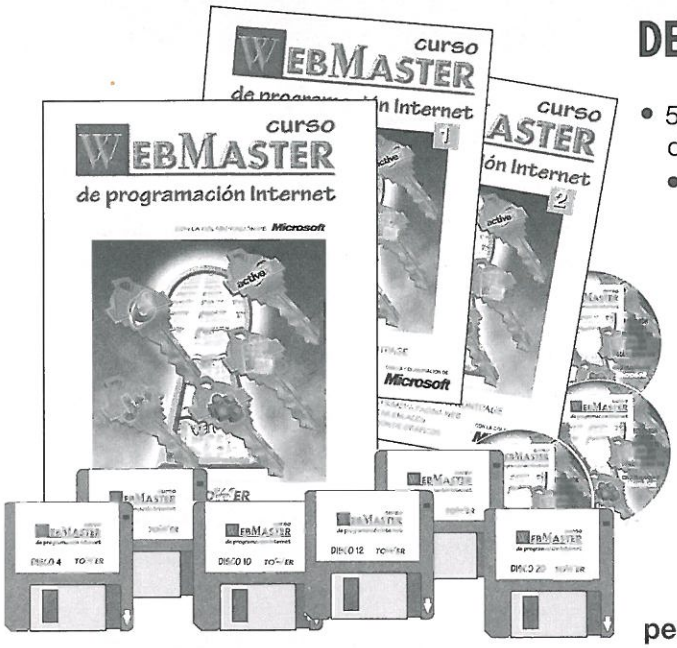
Una de las características esenciales que son resultado de este proyecto de ambas empresas multinacionales, es que los beneficios de esta maniobra incluirán actualizaciones de productos, entre otras ventajas.

Por otro lado contará con descuentos especiales destinados a desarrolladores, así como herramientas, numerosos cursos de aprendizaje y servicios de consulta gratuitos.

Acompañando a estos factores, Microsoft también ofrecerá descuentos para desarrolladores en su Office Developer Edition y de Visual Studio Professional Edition.

Active X, Animaciones, CGI, Correo electrónico, Diseño gráfico, FTP, FrontPage, Html, Html Dinámico, Java, Java Script, Navegadores, Visual J ++.

PREPÁRATE PARA SER EL MEJOR WEBMASTER



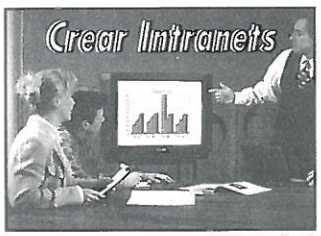
DESCRIPCIÓN DE LA COLECCIÓN:

- 5 CD ROM. Con los programas necesarios para el seguimiento del curso y el código de los ejercicios expuestos en el mismo.
- 500 fichas a todo color.
- 4 carpetas rígidas con anillas.
- 1 carpeta para los disquetes y CD-Roms.
- 48 disquetes.

Contenidos prácticos con ejemplos, ejercicios, casos, modelos, guías, etc...
Se entregará un diploma acreditativo a las personas que finalicen el curso.

Aprenderás a:

Valorado en 22.270 Ptas.



curso WEBMASTER de programación Internet

Por sólo 29.990 PTAS

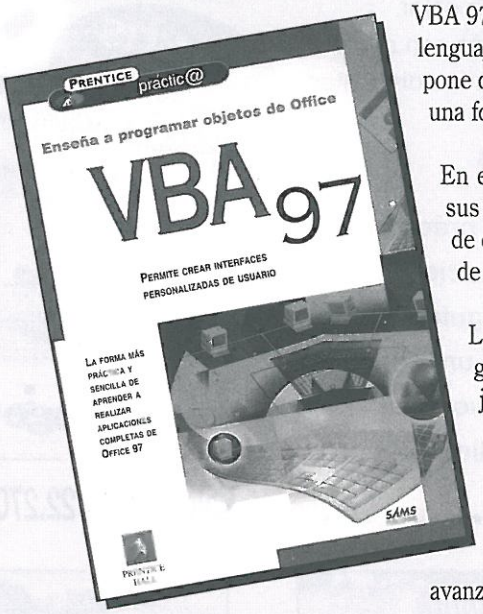
Solicítanos información en el tfno. 91-6614211, en el correo electrónico suscrip@towercom.es o envíanos este cupón y te contaremos cómo suscribirte y cómo funcionará la primera Bolsa de trabajo de expertos en Internet relacionada con este Curso.

Nombre y Apellidos ..... F. nacimiento .....
Domicilio ..... C.P. .....
Ciudad ..... Provincia ..... Telf.: .....
e-mail ..... Profesión .....
Apartado de correos ..... Código Postal .....

❑ Deseo recibir amplia información sobre el Curso Webmaster

LIBROS

Visual Basic para Aplicaciones 97



VBA 97 es la última versión del sistema de programación para aplicaciones basado en el lenguaje Visual Basic. Esta nueva versión simplifica aún más su utilización, ya que dispone de un sistema de ayuda mejorado y de un gran número de asistentes. Permite, de una forma práctica y sencilla, aprender a realizar aplicaciones completas de Office 97.

En este manual se estudian detalladamente todos los aspectos sobre cómo mejorar sus aplicaciones utilizando este lenguaje de programación, incluyendo gran número de ejemplos, de forma que el lector pueda seguir en todo momento las explicaciones de una forma sencilla y entretenida.

Los primeros capítulos del libro son una breve introducción a este lenguaje de programación, lo que permite aclarar conceptos relativos al funcionamiento y manejo de este nuevo entorno de desarrollo.

La segunda parte se centra sobre algunos aspectos relacionados con la creación de una interfaz personalizada de usuario. En este sentido, dedica este capítulo concreto al modo de llevar a cabo formularios y cuadros de diálogo, así como a la manipulación de eventos en los mismos y a sus tipos de controles. En partes más avanzadas del libro se detalla la programación en una base de datos VBA en Access.

Editorial: Prentice Hall
Nº de páginas: 187
Nivel: Intermedio

Autor: Paul McFredies
Idioma: Español
Precio: 2.490 Ptas. (I.V.A inc.)

Visual J++

Como respuesta al éxito de Java, Microsoft añadió este soporte a su navegador para la Web, Microsoft Internet Explorer, y se empezó a trabajar en el Visual J++, un entorno que saca a la programación en Java de la línea de órdenes y la empuja al mundo de interfaces gráficas de usuario al que pertenece. Los apartados más destacados de este libro son cuatro: Introducción, Programación, Introducción a Visual J++ y Programación con Visual J++.

El entorno de programación Visual J++ simplifica mucho el proceso de creación de aplicaciones elementales y aplicaciones independientes en Java, mientras que al mismo tiempo añade el poder de los trucos de Microsoft para acelerar el desarrollo del mismo lenguaje. Este entorno de programación es muy similar a Visual C++ y de hecho parecen prácticamente casi iguales en pantalla.

Esta obra trata aspectos relacionados con este código, las clases elementales de aplicaciones que se le atribuyen a este código, así como los tipos de utilidades y de red.

Editorial: Prentrice Hall
Nº de páginas: 637
Nivel: Experto

Autor: Mark Culverhouse, Clay Walnum y Nelson How
Idioma: Español
Precio: 6.900 Ptas. (I.V.A. inc.)

DIRECTX

Programación de DirectX con Delphi (IV)

Constantino Sánchez Ballesteros (constantino@redestb.es)

En este artículo se tratará de explicar la implementación de sonido a los juegos, utilizando el editor incluido en DGC para Delphi 3.0. Se mostrarán además los pasos necesarios para instalar DGC en las dos versiones conocidas en este momento.

Por fin tenemos en nuestras manos la versión de DGC para Delphi 3. El contenido de las librerías es el mismo que para Delphi 2.0, con la salvedad de que incorpora una serie de ficheros que efectuarán la instalación del paquete de software en Delphi 3. Al finalizar este artículo se os mostrarán los pasos necesarios a seguir, para instalar DGC en las dos versiones de Delphi.

Próximamente, los autores de DGC implementarán los modos *Hi-color* y *Windowed mode* (modo ventana) para incorporarlos a nuestros proyectos. Con *Hi-color* tendremos la posibilidad de utilizar miles de colores en los gráficos. *Windowed mode* permite que los juegos se ejecuten en una ventana normal de *Windows* en lugar de utilizar el modo a pantalla completa.

Editor de librerías de sonidos

Puesto que en este número vamos a abordar la implementación de sonido en nuestros juegos, debemos saber cómo utilizar el editor que viene incluido en DGC. El funcionamiento global es el mismo que el de la librería de imágenes, es decir, consiste en la recopilaci;on de una serie de archivos para guardarlos en un sólo fichero y en la posterior utilización por separado, cuando se considere necesario.

En esta imagen podéis observar el aspecto visual de la pantalla de presentación del recopilador de sonidos:

Observando la imagen, vemos varios apartados destinados a diferentes funciones. En la parte inferior, nos encontramos con tres casillas de notificación diferentes:

- Frequency: Va a mostrar la frecuencia a la que se encuentra grabado el sonido que tengamos activado en el editor.

El editor de sonidos facilitará el acceso y ordenación de los archivos WAV que utilicemos en nuestro video juego

- Channels: Si aparece el valor 1 en un sonido, esto quiere decir básicamente, que dicho sonido se ha grabado en Mono. En el caso de que aparezca el valor 2, es que se ha utilizado Estéreo.

- BPS: Se refiere a los Bits utilizados para representar el sonido. Por norma general, aparecerá un 8 para los grabados en Mono y 16 para los que utilicen Estéreo.

- Size: Informará del tamaño del sonido seleccionado.

Debajo del menú de la ventana, aparecen 3 iconos:

El primero de ellos se encarga de establecer una nueva librería de sonidos. El segundo abre una librería que tengamos guardada en el disco duro u otro soporte. El tercer icono guarda los datos de la librería de sonido actual.

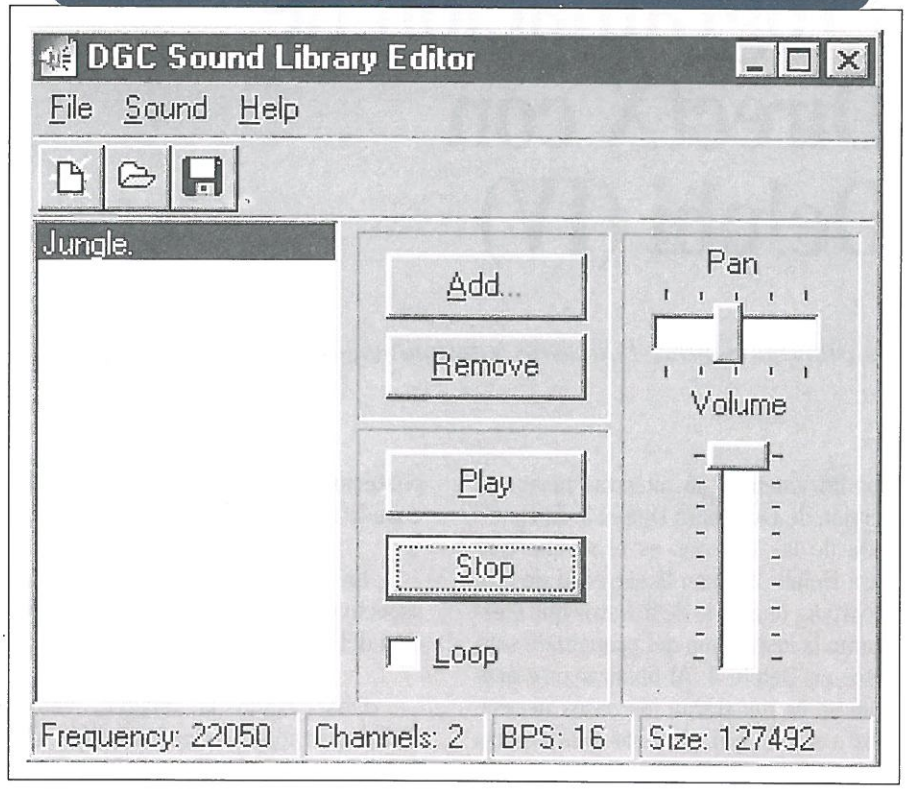
Lo primero que tenemos que hacer cuando queramos crear una librería, es abrir el programa (lógicamente) y pulsar el botón que lleva la etiqueta **ADD...** Esto abrirá inmediatamente una ventana con estructura de directorios que nos permitirá seleccionar un fichero de sonido (de tipo WAV) para incorporarlo al programa.

El botón con la etiqueta **Remove** se encarga de borrar el sonido seleccionado al ser pulsado. Con **Play** podemos reproducirlo y pararlo mediante **Stop**. La casilla de verificación **Loop** permite que el WAV sea reproducido continuamente.

A la derecha de la ventana se pueden ver 2 controles de barra deslizante:

- Pan: se encarga de efectuar el *panning* del sonido. Este efecto consiste en reproducir los datos de audio en una posición determinada del espacio tridimensional representado por los altavoces. Si movemos el **Pan** hacia la izquierda, el sonido se reproducirá en mayor cantidad sobre el altavoz izquierdo. De manera análoga, si deslizamos el control hacia la derecha, el sonido será reproducido por el altavoz derecho. Manteniendo centrado el **Pan**, se logra que los datos de audio sean distribuidos en su reproducción con un porcentaje del cincuenta por ciento para cada altavoz.

Detalle del editor de librerías de sonido.



- Volume: es el control encargado de seleccionar el volumen a utilizar en el sonido. Variando la posición del control modificaremos el nivel audible del sonido.

El menú del programa se encarga principalmente de la realización de varias de las tareas anteriormente mencionadas. Una vez hayamos logrado recopilar todos los sonidos a utilizar en nuestro videojuego, sólo debemos grabar la librería a un archivo. Con ello tendremos a nuestra disposición todo el soporte de audio para su reproducción. El archivo resultante llevará la extensión *SDL*.

- **TDGCSOUNDBUFFER**

En este apartado vamos a ver cada una de las propiedades y métodos que se pueden utilizar en la programación del sonido utilizando DGC. La clase *TDGCSoundBuffer* encapsula el objeto de DirectX "DirectSoundBuffer". Este permite la puesta en práctica de muchas funciones y propiedades que sirven para la asignación de opciones como volumen, *panning* y frecuencia.

Cuando se habla del término *sample*, nos estamos refiriendo al archivo de sonido utilizado:

- **Propiedades:**

- *BitsPerSample: Integer;*
  Número de bits per sample utilizados. Pueden ser valores de 8 y 16.

- *Channels: Integer;*
  Número de canales utilizados por el *sample*.

- *Frequency: Integer;*
  Se trata de la frecuencia del *sample*. Dependiendo de la calidad en el sonido, puede variar de 11000 a 44000.

- *Pan: Integer*;
  Esta propiedad controla desde dónde viene el sonido en relación con los altavoces. Un valor de 0 hace que el sonido venga de ambos altavoces por igual. Un valor negativo incrementa la reproducción sobre el altavoz izquierdo y se decrementa sobre el derecho. Con esta propiedad podemos crear sonido *'pseudo-posicional'* 3D.

- *Playing: Boolean*;
  En el caso de estar reproduciendo un sonido, esta propiedad retornará el valor *true*.

- *Position: Integer*;
  Si la posición es 0, entonces el sonido se encuentra al comienzo del muestreo. El método *Play* (el cual veremos más adelante) reproducirá el sonido desde esa posición. Si el método *Replay* es utilizado, la posición volverá a establecerse a 0 antes de que el sonido vuelva a ser reproducido.

- *Size: Integer*;
  Nos retorna el tamaño del archivo de sonido en *bytes*.

- *Volume: Integer*;
  Representa el volumen de sonido. Si cambiamos el valor del mismo durante la reproducción de los sonidos, podemos conseguir efectos bastante curiosos que pueden dar calidad al juego.

Con la clase TDGCSOUNDBUFFER tendremos acceso directo a todas las funcionalidades de sonido que puede ofrecer DirectSound

- **Métodos:**

- *Procedure Play*;
  Este método reproducirá el sonido desde la posición actual. Para reproducir el sonido siempre desde el principio, debemos utilizar *Replay*.

- *Procedure PlayLoop*;
  Reproducirá el sonido indefinidamente hasta que lo paremos con el procedimiento *Stop*.

- *Procedure Replay*;
  Este método parará cualquier sonido que se esté reproduciendo en un momento determinado, asignando automáticamente la propiedad *Position* a 0, y comenzará a reproducirlo íntegramente.

- *Procedure Stop*;
  Detendrá la reproducción del sonido actual.

- **TDGCAUDIO**

Para poder utilizar las propiedades y métodos de *TDGCSOUNDBUFFER*, es necesario enlazar en el formulario de nuestro programa un componente *TDGCAUDIO*.

Con *TDGCAudio*, inicializaremos *DirectSound* y cargaremos cualquier librería de sonidos que hayamos creado con nuestro editor. Este es, probablemente, el componente de DGC más sencillo pudiendo ser usado junto con cualquier aplicación regular de Delphi en modo ventana.

Este componente está integrado por dos propiedades:

- *Sound [Idx: Integer]: TDGCSoundBuffer*;
  La matriz establecida en la propiedad *Sound* retornará un sonido correspondiente a su posición en la librería de sonidos, y en el formato de un objeto *TDGCSoundBuffer*.
  El siguiente ejemplo tocará el primer sonido de la librería:

```
DGCAudio1.Sound[0].Play;
```

- *SoundLibrary: TDGCSoundLib*;
  Soundlibrary contiene un grupo de sonidos creados con el editor anteriormente comentado. Esta propiedad puede ser asignada en tiempo de diseño utilizando el inspector de objetos para enlazarlo a un componente *TDGCSoundLib*.

- **TDGCSOUNDLIB**

Este componente, básicamente es utilizado para guardar sonidos junto con el programa ejecutable. *TDGCSoundlib* es utilizado enlazándolo junto con la propiedad *Soundlibrary* de la clase *TDGCAudio*. Los sonidos son cargados dentro del componente utilizando el editor de propiedades.

- **TDGCSURFACE**

La clase TDGCSurface encapsula el objeto *DirectDrawSurface2* de DirectX y la clase estándar de Delphi *TCanvas*. Se han implementado funcionalidades adicionales para la detección de colisiones, *scrolling* y *clipping*. La mayoría de las funciones *Blit* de DirectDraw han sido simplificadas en esta clase y utilizan los métodos *GetPointer* y *ReleasePointer* para tener un fácil acceso a los *buffers* de memoria que utilicemos para implementar nuestras propias rutinas gráficas.

- **Propiedades**

- *Surface: IDirectDrawSurface2*;
  Esta propiedad de tipo "sólo lectura", retorna el objeto DirectDrawSurface2 utilizado por esta clase y se necesita si queremos llamar directamente a las funciones API de DirectDraw.

- *Canvas: TDGCCanvas*;
  La propiedad Canvas actúa justamente como la clase regular de Delphi TCanvas pero con una mayor diferencia. Un nuevo método ha sido introducido para liberar el contexto del dispositivo (DC) utilizado por el *buffer*. Este método es Release y debe ser llamado al final

de cada grupo de comandos *Tcanvas* que hayamos utilizado. Si el DC no es liberado, Windows llegará a colgarse.

El siguiente ejemplo dibujará un texto (**"Sólo Programadores"**) en el *Front-Buffer* (*buffer* primario):

```
with DGCScreen1.Front.Canvas do
begin
  Font.Color := clBlue;
  Brush.Color := clYellow;
  TextOut(0, 0, 'Solo Programadores');
  Release;
end;
```

- *TransparentColor: Byte;*
  TransparentColor define el índice de la paleta gráfica referente a los colores transparentes que se utilizarán en un *buffer*. Cierto número de métodos TDGCSurface tienen la opción de dibujar *buffers* de forma transparente, incluyendo entre ellos *Draw* y *Bltfast*.

- *Width: Integer;*
  Propiedad de sólo lectura que indica el ancho de un *buffer* gráfico en *píxels*.

- *Height: Integer;*
  Propiedad de sólo lectura que indica el alto de un buffer gráfico en *píxels*.

- *WidthBytes: Integer;*
  Este es el número de *bytes* que cada línea escaneada toma en memoria para los *bits* de un *buffer*. Si usamos *GetPointer* y *ReleasePointer* para actualizar el *buffer* de memoria directamente, debemos asignar este valor para calcular la distancia a la siguiente línea a escanear, y no utilizar la propiedad *Width*.

- *ClientRect: TRect;*
  Retorna el tamaño de un *buffer* en formato *TRect*.

- *ClipRect: TRect;*
  Asignará la región de *clipping* especificada para un *buffer* dado. El valor por defecto es todo el *buffer*. Actualmente, DGC sólo usa regiones de *clipping* con los comandos *Draw*, *Tile* y *Scroll*.

La clase TDGCSURFACE permite el acceso directo a la memoria gráfica implementado por DirectDraw

- **Métodos**

- *BltClip;*
  Procedure BltClip(X, Y: Integer; Source: TDGCSurface; SrcRect: TRect, Transparent: Boolean);

  Este es un método similar a *BltFast* que es utilizado para copiar una parte de un *buffer* a otro. La diferencia radica en que la copia será efectuada automáticamente con *clipping* usando los valores de *ClipRect*.

- *BltFast;*
  Procedure BltFast(X, Y: Integer; Source: TDGCSurface; SrcRect: TRect, Transparent: Boolean);

  BltFast dibujará parte del *buffer Source* en las posiciones **X**, **Y** utilizando *SrcRect*. Si *Transparent* toma el valor de *True*, el *buffer* es visualizado transparentemente. *Source* sólo será transparente si se ha asignado un valor a *TransparentColor*. Este método no soporta *clipping*.

  El siguiente ejemplo copiará el *Front-buffer* al *Back-buffer*, consiguiendo de esta manera que ambos puedan tener la misma información gráfica:

```
DGCScreen1.Back.BltFast(0, 0,
DGCScreen1.Front,
DGCScreen1.Front.ClientRect, False);
```

- *CollisionTest;*
  Function CollisionTest(X,Y: Integer;SourceSurface: TDGCSurface;SX,SY: Integer; PixelTest: Boolean): Boolean;

  Chequeará si la imagen contenida en este *buffer* en las posiciones **X** e **Y**, se solapa con la imagen contenida en *SourceSurface* con las posiciones **SX** y **SY**. Si hay colisión, la función retorna el valor *True*. Si el parámetro *PixelTest* es asignado como *false*, entonces una colisión es retornada si cualquier región de las dos imágenes se solapa entre sí (en este caso, la función de Windows *IntersectRect* es llamada). Si *PixelTest* toma el valor de *True*, las regiones solapadas entre imágenes son chequeadas para colores no transparentes en la misma posición de cada una de las imágenes. Si *TransparentColor* no ha sido asignada, el valor establecido por defecto es 0.

  El siguiente ejemplo efectúa un test de *píxels* entre dos imágenes mediante los grăficos contenidos en *image[0]* e *image[1]*. Si las imágenes colisionan, la rutina *PlayerDead* es invocada:

```
if DGCScreen1.Images[0].CollisionTest(AlienX,
AlienY, DGCScreen1.Images[1], PlayerX, PlayerY,
True) then begin

  PlayerDead;
end;
```

- *Create;*
  constructor Create(DirectDraw2: IDirectDraw2; w, h: Integer);

  El constructor de TDGCSurface toma tres parámetros: el objeto *DirectDraw2* y *width - height*. *DirectDraw2* es una propiedad de TDGCScreen. Una forma mejor de crear nuestros propios objetos de tipo *buffer* es pasar un objeto

TDGCSurface no inicializado al método *CreateSurface* de TDGCScreen.

- *Draw*;
  Procedure Draw(X, Y: Integer; SourceSurface: TDGCSurface; Transparent: Boolean;

  El método *Draw* dibujará el buffer SourceSurface en las posiciones X, Y; Si *Transparent* vale *True*, el *buffer* se visualizará transparente.
  El siguiente ejemplo, dibujará la primera imagen de la librería accedida mediante *image (0)* en la posición 50, 50 sobre el *backbuffer*, utilizando transparencia:

```
DGCScreen1.Back.Draw(50, 50,
DGCScreen1.Images[0], True);
```

- *Erase*:
  Procedure Erase(Color: Byte);

  Este método borrará un *buffer* entero con el color especificado. El color está formado mediante paleta indexada.

- *Eraserect;*
  Procedure Erase(TheRect: TRect; Color: Byte);

  Este método se encarga de borrar un rectángulo definido en un *buffer* con el color especificado. Para borrar completamente el *buffer*, es mejor utilizar el método *Erase*.

- *Getpointer*:
  Function GetPointer: Pointer;

  GetPointer tomará un puntero en memoria, conteniendo los *bits* de un *buffer*. Es necesario hacer la llamada a *ReleasePointer* después de su uso. Windows estará protegido entre las funciones utilizadas con *GetPointer/ ReleasePointer* para evitar errores de excepción típicos del S.O.

- *ReleasePointer*:
  Procedure ReleasePointer;

  Como hemos visto en el método anterior, se encarga de liberar el espacio obtenido por *GetPointer*.

- *Scroll*:
  Procedure Scroll(XAmount, YAmount: Integer);

  Este método efectuará un *scroll* del rectángulo obtenido en sus variables. *XAmount* e *YAmount* pueden ser positivos o negativos. Solamente la región asignada por *ClipRect* podrá ser utilizada para el *scroll*. El valor por defecto de *ClipRect* es el *buffer* entero.

- *Setpixel*:
  Procedure SetPixel(X, Y: Integer; Color: Byte;

  Se encarga de poner un *píxel* en las coordenadas **X, Y** utilizando el color asignado en *Color*. Este método es efectuado protegiendo el buffer de memoria con *GetPointer*, asignando el apropiado *byte* en memoria y liberando el espacio utilizado mediante el método *ReleasePointer*. No tiene implementado *clipping*, por lo tanto, si se dibuja un *píxel* fuera del área del *buffer*, pueden ocurrir efectos impredecibles con Windows.

- *StretchDraw*;
  Procedure StretchDraw(TheRect: TRect; SourceSurface: TDGCSurface; Transparent: Boolean);

  StretchDraw dibujará el gráfico contenido en *SourceSurface* en el rectángulo especificado, efectuando *stretching* de la imagen de forma acorde a los parámetros pasados en *TheRect*. No se ha implementado *clipping*, por lo tanto, no debemos poner el *sprite* fuera del área cliente.

- *StretchFlip*:
  Procedure StretchFlip(TheRect: TRect; SourceSurface: TDGCSurface; Transparent, FlipX, FlipY: Boolean;

  Este método es el mismo que *StretchDraw* pero con unos cuantos parámetros más. Si los parámetros *FlipX* y *FlipY* obtienen el valor de *true*, la imagen será copiada con *flipping* (dada la vuelta).

- *Tile*:
  Procedure Tile(X, Y: Integer; SourceSurface: TDGCSurface; Transparent: Boolean;

  Tile dibujará de forma repetida el *buffer SourceSurface* en las posiciones **X, Y** hasta que el rectángulo destino del mismo se haya rellenado. La posición 0, 0 es la esquina superior izquierda de la región del rectángulo destino, no del *screen*.

- **Eventos**

- *Property OnSurfaceLost: TNotifyEvent;*
  Este evento es llamado cuando DERR_SURFACELOST es retornado por alguna función. El *buffer* de memoria se puede perder cuando una aplicación es cambiada utilizando ALT+TAB. Si este hecho ocurre, el *buffer* debe ser restaurado y dibujado de nuevo.

Programación de animación y sonido

En el ejemplo de este mes, vamos a poner en práctica la teoría de algunas de las propiedades y métodos representados anteriormente. Las propiedades y métodos relacionados con el componente TDGCSpriteMgr serán revisados cuando avancemos en el curso, aunque en este ejemplo comencemos a utilizar alguna instrucción. La funcionalidad del programa representa la utilización de *sprites* animados juntos con soporte de sonido y colisiones. He creado una animación para el *sprite* definido como "invasor", el cual posee 2 imágenes en el banco correspondiente para crear la sensación de estar animado. La nave contiene un solo *sprite*.

Con los cursores podremos mover la nave de un lado a otro de la pantalla. Si hay colisión entre ambos *sprites*, la nave será sustituida por una animación de explosión.

A continuación, os expongo los extractos más interesantes del programa que podréis encontrar por completo en el CD de la revista:

```
procedure TForm1.DGCScreen1Initialize(Sender:
TObject);
begin
```

Dibuja en el *backbuffer*

```
DrawFrame;
```

Efectúa un fundido de la paleta gráfica tranformándola al color negro. De esta manera se logra oscurecer completamente la pantalla

```
DGCScreen1.FadePaletteOut(1);
```

Realiza el volcado del *backbuffer* al *frontbuffer* para que los gráficos sean visualizados, aunque todavía no los podremos ver puesto que la paleta está completamente de colores negros.

```
DGCScreen1.Flip;
```

Finalmente, restauramos suavemente la paleta gráfica con *FadepaletteIn*.

```
DGCScreen1.FadePaletteIn(100);
```

Creamos una copia del *frontbuffer* en el *backbuffer* para que los dos contengan la misma información.

```
DGCScreen1.Back.BltFast(0, 0,
DGCScreen1.Front,
DGCScreen1.Front.ClientRect, False);
```

En este punto es donde creamos las animaciones. Hay que constar que aquí sólo definimos las animaciones como tal, y no los *sprites* como alguno pudiera pensar. El nombre de la animación es opcional (nosotros lo llamaremos invasor), y el segundo parámetro se refiere al *flag* Loopanimation.

```
DGCSpriteMgr1.Animations.Add('Invasor',
True);
```

El siguiente paso consiste en incorporar imágenes a las animaciones que hayan sido creadas con anterioridad. Si queremos utilizar las mismas imágenes para las 32 direcciones posibles de rotación, usaremos el método *SetAllDirections*. El valor 100 se refiere al parámetro velocidad. [1, 2] son los números representativos de las dos imágenes que van a ser utilizadas para crear la animación.

```
DGCSpriteMgr1.Animations[0].SetAllDirections(10
0, [1, 2]);
```

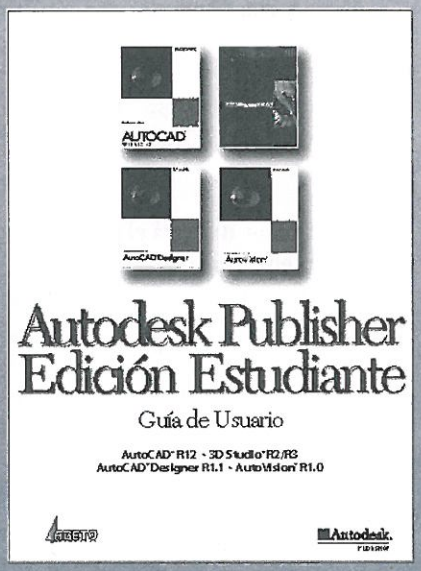

AUTODESK PUBLISHER EDICIÓN ESTUDIANTE

¿QUÉ PROGRAMAS INCLUYE?

- AutoCAD R12
- 3D Studio R2/R3
- AutoCAD Designer R1.1
- Autovision R1.0.

Y además:

Más de 700 páginas de lecciones de Autodesk con guías de referencia y documentación software on-line



Formato 21x29,7

TODO LO QUE NECESITA PARA E DISEÑO Y VISUALIZACIÓN PROFESIONAL EN 2D Y 3D LO HALLARÁ EN EST PAQUETE DE PROGRAMAS.

EDICIÓN ESPECIAL PARA ESTUDIANTES
19.500 PTAS. (I.V.A. INCLUIDO)

DE VENTA EN LIBRERÍAS Y DISTRIBUIDORES AUTODESK

Definimos las animaciones correspondientes a la nave y la explosión. Puesto que vamos a utilizar la misma imagen para las 32 direcciones de rotación posibles, asignaremos el método *SetAllDirections*.

```
DGCSpriteMgr1.Animations.Add('nave', True);

DGCSpriteMgr1.Animations[1].SetAllDirections(10
0, [0]);
```

Para la explosión, ponemos el *flag* de animación en *false* puesto que no se repetirá continuamente, y cuando la explosión acabe, debe desaparecer de la pantalla. Los *frames* correspondientes a la explosión están asignados en el banco de imágenes como [3, 4, 5, 6].

```
  DGCSpriteMgr1.Animations.Add('Explode',
False);

DGCSpriteMgr1.Animations[2].SetAllDirections(10
0, [3, 4, 5, 6]);
```

Ahora que hemos definido todas las animaciones posibles en el juego, es hora de crear los *sprites* correspondientes. Comenzamos con la nave controlada por el jugador. Las propiedades de tipo *Allow* son utilizadas para que la nave sólo pueda moverse de izquierda a derecha.

```
DGCSpriteMgr1.Sprites.AddPlayer8(0, 320, 440, 0,
3, 1);
  with DGCSpriteMgr1.Sprites[0] as
TDGCPlayer8 do
  begin
```

Sin movimiento hacia arriba.

```
AllowUp := False;
```

Sin movimiento hacia abajo.

```
AllowDown := False;
```

Aceleración de la nave.

```
Acceleration := 0.1;
```

Deceleración de la nave.

```
Decceleration := 0.1;
```

Cada *sprite* puede moverse con un área definida por un rectángulo imaginario establecido por la propiedad *Limits*. Cuando el *sprite* llega a la esquina de cualquier parte del rectángulo, ocurre un evento. Por ejemplo, cuando el *sprite* llegue a uno de los lados, podemos hacer que la nave rebote hacia el lado contrario, o por el contrario, que se pare. Solamente una acción puede definirse por cada cara de los límites imaginarios del rectángulo. Si queremos la misma acción para todas las caras, usaremos la propiedad *Allactions*. Para asignar eventos sobre caras determinadas, debemos usar las propiedades *LeftAction*, *TopAction*, *RightAction* y *BottomAction* correspondientes a las partes izquierda, arriba, derecha y abajo del rectángulo respectivamente.

Una vez definidas las animaciones de un juego se crean los sprites que le corresponden

laWrap absorverá el *sprite* de un lado a otro de la pantalla.

```
AllActions := laWrap;
end;
```

Crea el *sprite* del invasor (TDGCBouncer)

```
  DGCSpriteMgr1.Sprites.AddBouncer(0, 90,
300, 14, 5, 0);
  with DGCSpriteMgr1.Sprites[1] as
TDGCBouncer do
  begin
    Limits := Rect(0, 200, 640, 468);
```

El invasor aumentará su velocidad.

```
    Acceleration := 0.01;
```

Creará un rebote del *sprite* invasor.

```
    AllActions := laBounce;
  end;
```

Asignamos la región de *clipping* en el área en la que estamos dibujando.

```
DGCScreen1.ClipRect := Rect(0, 200, 640, 468);
TraceString('Inicialización');
```

Activamos el *page flipping* para el volcado de *buffers*.

```
  DGCScreen1.FlippingEnabled := True;
end;


procedure TForm1.DGCScreen1Flip(Sender:
TObject);
var
  ShipHit: Boolean;
begin
```

Borra el área sobre el que se moverán los *sprites*. Esta región es considerada como región de *clipping*:

```
DGCScreen1.Back.EraseRect(Rect(0, 200, 640,
468), 0);
```

Dibuja los *sprites*.

```
DGCSpriteMgr1.Draw;
```

Actualiza los *sprites* en sus nuevas posiciones.

```
DGCSpriteMgr1.Update;
```

Chequeo de colisión entre el sprite 0 y el 1 (el invasor y la nave)..

```
if (DGCSpriteMgr1.Sprites.Collision(0, 1)) and
   (DGCSpriteMgr1.Sprites[0].Animation <> 2)
then
  begin
```

En el supuesto de que se produzca cualquier tipo de colisión, hacemos que la nave explote...

```
  with DGCSpriteMgr1.Sprites[0] as
TDGCPlayer8 do
  begin
        ActiveAnimation := 2;
   IdleAnimation := 2;
```

Desactiva el chequeo automático del teclado.

```
Automatic := False;
      end;
```

Reproducimos el sonido de explosión con *Replay*.

```
    DGCAudio1.Sound[0].Replay;
```

A la vez que explota nuestra nave, hacemos que el invasor desaparezca utilizando una dirección de tipo *Random*. La propiedad *AllActions* es asignada como *laStopOutside*, lo cual significa que el invasor parará de moverse cuando haya salido por completo fuera de la pantalla.

```
      with DGCSpriteMgr1.Sprites[1] as
TDGCBouncer do
      begin
       Randomize;
       Direction := Random(32);
```

Asigna la máxima velocidad.

```
       MaxSpeed := 5;
```

Asigna la velocidad actual.

```
       Speed := 5;
 AllActions := laStopOutSide;
      end;
   end;
end;

procedure
TForm1.DGCSpriteMgr1AnimationEnd(Sprite:
TDGCSprite);
begin
```

En este programa en particular, este evento es llamado cuando la animación de explosión ha llegado a su fin. El evento *OnAnimationEnd* sólo es llamado si la animación no se reproduce de forma infinita.

```
    DGCSpriteMgr1.Sprites[0].Hide;
    DGCSpriteMgr1.Sprites[0].Disable;
end;

procedure
TForm1.DGCSpriteMgr1SpriteDirChange(Sprite:
TDGCSprite;
  LimitsSide: TLimitsSide);
begin
```

Reproducimos un sonido que suena como un *"boing!"* cuando el invasor golpea los límites del rectángulo definido para sus límites de movimiento.

```
 if Sprite is TDGCBouncer then
      DGCAudio1.Sound[1].Replay;
end;
```

Instalación de DGC sobre Delphi 2.0 y 3.0

Vamos a abordar la instalación de DGC sobre estas dos versiones de Delphi. Los pasos siguientes corresponden a la instalación sobre la versión 2.0:

Antes de instalar los componentes de DGC, debemos verificar las siguientes condiciones:

1. No tener instalada ninguna versión previa de DGC.
2. No tener ninguna otra versión diferente de los *headers* de DirectX a los incluídos por **BlakeStone** en DGC (ddraw.pas, dsound.pas etc.)
3. Tener instalado en nuestro sistema DirectX 3 o superior.
4. Es conveniente realizar una copia de seguridad del fichero estándar de Delphi "cmplib32.dcl" localizado en el directorio **Bin**. Si al realizar la instalación de DGC tenemos algún problema, podremos restaurar los parámetros copiando este fichero a su lugar de origen (directorio **Bin**).

- **Instalando los componentes**

1. Descomprimimos el fichero *dgcbeta6.zip* en un directorio temporal y ejecutamos el programa *setup.exe*.
2. Seleccionamos la opción del menú *'Install"* desde el menú *'Component'*.
3. Agregamos ene este momento el fichero "dgcreg.pas".
4. Pulsamos OK para construir la librería de componentes.
5. Si los componentes han sido instalados correctamente, una pestaña con la nota DGC debería aparecer en la paleta de componentes. Debemos asegurarnos de tener desactivada la opción **'Break on Exception'** localizada en el menú *Tools/Options*.

Para instalar los componentes hay que verificar una serie de condiciones, tales como no tener ninguna versión previa de DGC

Un error "1157" durante la instalación de los componentes, significa que no tenemos instalado DirectX 3 o superior.

Las siguientes líneas se refieren a la instalación de DGC sobre Delphi 3.0:

1. Como ocurre en la versión anterior, descomprimimos el fichero *dgcbeta6.zip* en un directorio temporal y ejecutamos el programa *setup.exe*.
2. Descomprimimos todos los ficheros contenidos en *dgcb6p.zip* dentro del directorio **Bin** de DGC.
3. Ejecutamos Delphi 3 y elegimos la opción perteneciente al menú *Component/Install Packages*.
4. Cuando aparezca el cuadro de diálogo, pulsamos sobre el botón Add y seleccionamos el fichero *DGCBETA6.DPL* contenido en el directorio **Bin** de DGC.
5. Pulsamos OK del cuadro de diálogo para completar la instalación.
6. Llegados a este punto deberíamos tener una pestaña con el nombre DGC en la paleta de componentes.

Creación de controles ActiveX (I)

Jordi Agost Moré – agost@eup.udl.es

Empezaremos aclarando la terminología que se empleará a lo largo del artículo.

Lo primero que tenemos que tener en cuenta es que cuando trabajemos con controles, nuestro enfoque va a cambiar un poco. Hasta ahora habíamos creado clases, documentos ActiveX realizábamos nuestro propio código y procedimientos, lo ejecutábamos y observábamos el resultado. Con los controles ActiveX tenemos que tener en cuenta que no solamente vamos a tener código sino que ahora también vamos a tener componentes visuales.

Por otro lado un control ActiveX no puede existir simplemente por si mismo, necesita un contenedor (tal y como sucedía cuando veíamos los documentos ActiveX). Además otra diferencia es que los controles ActiveX son utilizados tanto por usuarios finales como programadores, desde de puntos de vista diferentes.

Otro factor que debemos tener en cuenta es que cuando desarrollemos un control ActiveX, no lo haremos de forma individual, sino que en realidad estaremos creando lo que llamamos una *clase de control*, una clase genérica a partir de la cual iremos creando instancias. Cada vez que coloquemos un control dentro de un formulario (todos los controles necesitan tener un contenedor, en nuestro caso el formulario), estaremos creando una instancia de la clase de control. Para entender el correcto funcionamiento de éste, debemos tener presente, que la instancia será distinta de la clase de control que creemos, como veremos en el artículo.

Una cuestión importante, es que los controles van a ser para nosotros objetos proporcionados por los componentes. Un componente de control dará lugar a diferentes tipos. Los archivos .ocx son los componentes de control, que como ya hemos dicho nos proporcionarán las clases. Dichas clases de control vienen definidas por ser archivos con la extensión .ctl. Cuando generemos el proyecto, Visual Basic asignará a cada componente de control la extensión .ocx. Entonces, una vez generado el proyecto, si incorporamos el componente de control a un proyecto normal podremos usar cualquier control que nosotros hayamos definido dentro del componente de control.

Es necesaria tener muy clara la terminología sobre los creadores y los usuarios del control. Por ejemplo, si nosotros desarrollamos un control, y posteriormente lo compilamos como un componente de control o archivo .ocx, podemos decir con toda propiedad que seremos el autor del mismo. En el caso de utilizar el control en un proyecto, seremos programadores que estemos usando aquel control determinado, y que en este caso coincide con el autor. Pero podría pasar que cuando hayamos desarrollado nuestro control, lo pongamos a la venta y un programador lo compre para incorporarlo a su proyecto. Finalmente puede ser

Desde que empezamos a desempeñar tareas con Visual Basic 5.0, estamos trabajando con controles. Por ejemplo al diseñar un formulario, ponemos una caja de texto junto con una etiqueta. En este caso estamos insertando en nuestro proyecto controles ActiveX. El artículo expondrá cómo crear controles ActiveX desde VB.

que dicho programador venda su proyecto a un usuario final, que instalará y usará la aplicación. También es posible que el usuario compre un determinado control, y lo use para incorporarlo en otras aplicaciones (no tiene que ser un experto programador para realizar dicho paso), tales como Microsoft Office.

Cómo crear un control ActiveX

Vamos a empezar ahora nuestra creación del control ActiveX, en el menú Archivo elegimos Nuevo Proyecto y hacemos clic en el icono de Control ActiveX. En este momento Visual Basic añadirá automáticamente al proyecto un diseñador UserControl. Cualquiera que sea el control que estemos creando, éste se constituirá siempre por un objeto llamado UserControl y otros controles denominados *controles constituyentes* que se colocarán en él. Los controles constituyentes se pondrán en el diseñador del objeto UserControl de igual forma que se colocan los controles en un formulario. Y además al igual que los formularios los objetos de este tipo tienen código y un aspecto visual. Igualmente de la misma forma que los formularios, dichos objetos se almacenan en archivos de texto normal, con el código que le hayamos introducido y los valores de las propiedades del objeto mismo (del mismo modo que VB escribe un formulario con la extensión .frm). La extensión elegida, tal y como hemos dicho en el párrafo anterior, es .ctl. También en los formularios, si los objetos UserControl, poseen elementos gráficos como archivos de mapas de bits que no puedan almacenarse como texto; los guardará con la extensión .ctx (recordemos que para un formulario dicha extensión es .frx).

En la imagen se puede observar el aspecto del objeto UserControl. Si nos fijamos bien veremos que los controladores de arrastre (las figuras en forma de cuadrado que nos sirven para cambiar el tamaño del control), situados en la parte superior e izquierda del objeto UserControl están desactivados. Esto se debe a que los controles de usuario, deben estar siempre en la esquina superior izquierda por lo que no podemos desplazarlos. En cambio, los que si que podemos desplazar son los controladores de arrastre situados en la parte inferior y en el lado derecho del control. Siguiendo con el análisis de la imagen veremos que en la barra de título, o caption, nos muestra el nombre de la nueva clase de control que hemos creado: UserControl1.

Es posible agregar propiedades al control que se haya creado, de la misma manera que se añadían a las clases ActiveX

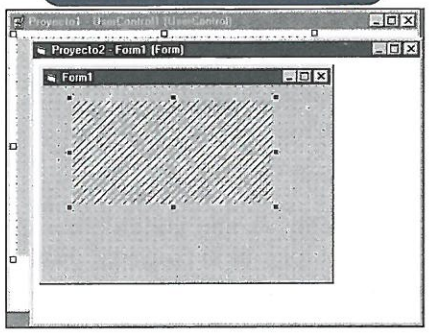
En la ventana de propiedades haciendo un doble clic sobre la propiedad Name, vamos a cambiar el nombre a *EjemploSP*, y observamos a la vez como el nombre de la barra de título del diseñador también cambia. La denominación introducida será la que se convertirá en el nombre de la clase de control. Por ejemplo TextBox es el nombre de la clase de control de las cajas de Texto, y para el programador que utilice nuestro control, *EjemploSP*, será el nombre genérico de la clase de control. Acto seguido guardamos el proyecto. Observemos que a la hora de guardar el proyecto, éste se almacenará con la extensión vbp (Visual Basic Project) y el control con la extensión .ctl.

Una instancia de un control no puede existir por si misma como sucede con los formularios. Al igual que los documentos ActiveX, debemos colocarla en un contenedor, el ejemplo más común de contenedor son los formularios de Visual Basic. Cuando coloquemos una instancia

de control en un formulario de Visual Basic, diremos que estamos ubicando el control ActiveX dentro de un contenedor, el formulario. A partir del momento en que esto haya sucedido los eventos quedarán disponibles en la ventana de formulario del control, y además por el hecho de estar en un contenedor, también tendremos acceso a otros servicios que proporciona el contenedor de los controles.

Así para probar nuestro control ActiveX necesitamos un contenedor. En nuestro caso el contenedor va a ser un formulario de prueba, aunque necesitamos algo más que añadir simplemente el formulario a nuestro proyecto y ejecutarlo. Vamos al menú Archivo y hacemos clic en Agregar Proyecto, y dentro del cuadro de diálogo elegimos la opción de un exe estándar. Si nos fijamos en el explorador de proyectos, veremos que se ha añadido uno nuevo que será el que por defecto se ejecutará cuando pulsemos F5 (tecla que ejecuta el proyecto actual). Esto sucede porque un proyecto de control ActiveX no puede ser nunca el proyecto inicial. Visual Basic nos indica este hecho poniendo el nombre del proyecto en negrita.

Ahora en el menú de archivo elegimos guardar grupo de proyectos, con lo que se nos guardará aquél que ha sido creado, apareciendo a su vez una nueva extensión: .vbg, extensión que se referirá al fichero que guarda información sobre el grupo de proyectos. Cuando vayamos a escribir el código tenemos que tener en cuenta en que momento se va a ejecutar éste, es decir, los controles



van a tener un co mportamiento determinado en tiempo de diseño (para el que escribiremos un código concreto) y otro comportamiento diferente en tiempo de ejecución. No obstante, habrá porciones de código que serán comunes a los dos modos, como por ejemplo el evento Resize del control (EjemploSP_Resize) que se ejecutará tanto en modo de diseño, como de ejecución. Para ver esto activamos nuestro control EjemploSP y le añadimos en el procedimiento Resize el siguiente código:

```
Private Sub UserControl_Resize()

  Static tmp%
  tmp% = tmp% + 1
  Debug.Print "Cambio de tamaño: " & tmp%

End Sub
```

Ahora con la finalidad de probar el control, tenemos que ejecutarlo, mientras el otro proyecto (que va a ser el contenedor del control) queda en modo de diseño. A continuación activamos la ventana del diseñador del control ActiveX y luego la cerramos presionando Control + F4. Al cerrar la ventana, EjemploSP se pone en modo de ejecución, y el icono que en la barra de herramientas apareció para él (es un icono general para cuando se añade un control de usuario) se activa. Tomamos el formulario del otro proyecto y le añadimos un control EjemploSP, automáticamente en la ventana de depuración aparece el siguiente mensaje: Cambio de tamaño: 1

Y tantas veces como cambiemos de tamaño irá apareciendo el mensaje (incrementándose el contador). Además en el formulario de prueba aparece el nombre de EjemploSP1 con lo que el "1" nos indica que hemos creado una instancia de la clase de control.

Si ahora abrimos el control ActiveX, y mientras lo visualizamos observamos la ventana del formulario, veremos que el sitio donde se encontraba el control está marcado con unas rayas diagonales (tal y como se ve en el gráfico) para así indicar que está inactivo. Al ponerse inactivo el control, también lo hará cualquier instancia que hayamos creado a partir de él. Si cambiamos el tamaño en la ventana de Inmediato no aparece ningún mensaje, ya que estamos en modo de diseño del control. Por lo tanto éste permanece inactivo.

Tiempo de vida de un control

A fin de lograr un dominio sobre la creación de los controles ActiveX, una de las primeras cuestiones que debemos conocer bien es su ciclo de vida, es decir conocer los eventos que se producen con la creación y la destrucción de un control. Vamos a observar cómo funcionan los eventos Initialize, InitProperties, ReadProperties, WriteProperties y Terminate. Abrimos la ventana de código del control ActiveX y escribimos el código en los correspondientes eventos:

```
Private Sub UserControl_Initialize()
  Debug.Print "Evento Initialize"
End Sub

Private Sub UserControl_InitProperties()
  Debug.Print "Evento InitProperties"
End Sub

Private Sub UserControl_ReadProperties(PropBag
      As PropertyBag)
  Debug.Print "Evento ReadProperties"
End Sub

Private Sub UserControl_Terminate()
  Debug.Print "Evento Terminate"
End Sub

Private Sub UserControl_WriteProperties(PropBag
      As PropertyBag)
  Debug.Print "Evento WriteProperties"
End Sub
```

Llevamos el diseñador del control adelante y lo cerramos con Ctrol + F4 (con lo que se ejecutará el control). Luego llevamos delante el formulario y pulsamos ctrol f4 para ejecutar el control, dirigiéndonos después al formulario. Observamos que aparece en la ventana de depuración:

```
Evento Initialize
Evento ReadProperties
Cambio de tamaño: 1
```

Esto sucede aún cuando no hayamos puesto ningún control en el formulario. Se debe a que cuando hemos cerrado el diseñador, la instancia se ha destruido y se ha vuelto a crear cuando se nos ha mostrado el formulario, y no se ha agregado un evento terminate (sería lógico si la instancia se ha destruido al cerrar el diseñador). Esto sucede porque el código del evento Terminate se ha añadido después de crear la instancia.

El punto anterior es sumamente importante porque nos proporciona una amplia visión sobre los diferentes puntos de vista que intervienen en el mundo de los controles. Si un programador añade un control a un formulario, puede pensar que es algo estático, que está ahí desde que lo añadió, pero para el programador de controles ActiveX, éstos se crean y destruyen automáticamente.

Limpiamos en este momento la ventana de depuracióncon el propósito de ejecutar todo el proyecto.

```
Evento WriteProperties
Evento Terminate
Evento Initialize
Evento ReadProperties
Cambio de tamaño: 1
```

El evento WriteProperties nos dará la oportunidad de escribir las propiedades que el desarrollador haya establecido en tiempo de diseño. Terminate destruye el control de modo de diseño. Posteriormente el evento Initialize crea el control, ReadProperties lee las propiedades y se produce el cambio de tamaño.

Si ahora añadimos un segundo control ActiveX a nuestro formulario contenedor veremos en la ventana de inmediato:

```
Evento Initialize
Cambio de tamaño: 1
Evento InitProperties
Cambio de tamaño: 2
```

El nuevo evento, se produce cada vez que agregamos a un formulario una nueva instancia de un control, en el que se establecen los valores determinados de las propiedades. En dicho evento podemos escribir código para establecer los valores predeterminados de cada valor de las diferentes propiedades del control; o simplemente realizar unas tareas concretas, siempre que el usuario cree este tipo de controles.

Dibujar el control

Para dibujar el control podemos utilizar métodos gráficos como por ejemplo Line, pero también podemos crear el aspecto visual a partir de controles ActiveX existentes (recordemos que en este caso, dichos controles los llamaremos controles constituyentes).

Podemos crear para nuestro ejemplo, una etiqueta, con la propiedad Name igual a Lab y una caja de texto, con la propiedad Name=Text. Aunque dicho control únicamente lo utilizaremos para crear nuestro ejemplo, también lo podríamos utilizar para crear la base, que por ejemplo, nos podría servir para hacer un contador. Por ejemplo si tenemos una serie de procesos que pueden tardar en llevarse a cabo, podría ser útil mostrar dicho control con un mensaje de que es lo que nuestro programa está haciendo, y el proceso en el que se encuentra.

Las propiedades

A nuestro control también le podemos agregar propiedades, de igual forma que lo hacíamos con las clases, tal y como vimos en un artículo anterior. Como ejemplo supongamos que deseamos crear una propiedad Titulo, que constituiría el mensaje que leería el usuario.

A tal efecto, seleccionamos nuestro control en el explorador de proyectos, lo visualizamos, hacemos doble clic para ver el código y en el menú Herramientas hacemos un clic en Agregar Procedimiento. En el cuadro Nombre escribimos Titulo, hacemos clic en Propiedad y Público. VB automáticamente nos creará unos procedimientos de propiedad, cambiaremos estos por el siguiente código:

```
Public Property Get Titulo() As String
  Titulo = LabelPrueba.Caption
End Property

Public Property Let Titulo(ByVal NewTitulo As
      String)
  LabelPrueba.Caption = NewTitulo
  PropertyChanged "Titulo"
End Property
```

Recordemos que el procedimiento Let se ejecutará cada vez que le asignemos un nuevo valor a la propiedad Titulo. Los procedimientos Let, además deben llamar a PropertyChanged, así logran indicar a la ventana de propiedades que actualice su presentación e informará a Visual Basic de que el procedimiento ha cambiado.

El procedimiento Get, por su parte se ejecutará siempre que le asignemos un nuevo valor a la propiedad Título.

Si queremos establecer una propiedad por defecto a nuestro control, este será el texto que mostrará por defecto. Tendríamos que añadir el siguiente código al procedimiento de evento UserControl_InitProperties

```
Private Sub UserControl_InitProperties()
  Debug.Print "Evento InitProperties"
  Titulo = "Prueba Sólo Programadares"
End Sub
```

Si ahora lo que pretendemos es guardar el valor de la propiedad Titulo, lo haríamos con el siguiente código:

```
Private Sub UserControl_WriteProperties(PropBag
      As PropertyBag)
  Debug.Print "Evento WriteProperties"
  PropBag.WriteProperty "Titulo", LabelPrueba,
      "Prueba Solo Programadores"
End Sub
```

Donde PropertyBag es como una bolsa donde guardaremos las propiedades de las propiedades. Esta bolsa será proporcionada por el contenedor, es como una especie de portapapeles que posee el contenedor, aunque al contrario que éste no podemos ver su contenido, ni sabemos como se guarda. Lo único que podemos hacer es poner valores dentro de esta "bolsa" y sacarlos de la misma. El primer argumento es el nombre de la propiedad que se usará como clave a la hora de recuperar los datos, y debemos utilizar aquí el nombre de la propiedad. El segundo argumento es el valor que guarda como información de tipo Variant. Y el tercer argumento es el valor predeterminado de la propiedad (VB para ahorrarse espacio a la hora de escribir el formulario en disco, mira si la propiedad es igual al valor predeterminado, si es así no la escribe con el consecuente ahorro en el tamaño de los formularios). El código para leer las propiedades sería:

```
Private Sub UserControl_ReadProperties(PropBag
      As PropertyBag)
  Debug.Print "Evento ReadProperties"
  LabelPrueba = PropBab.ReadProperty("Titulo",
      "Prueba Solo Programadores")
End Sub
```

El primer argumento será la clave a recuperar, y el segundo sería cierto

valor predeterminado, que se usará si no se ha guardado ningún valor, o si el usuario ha eliminado la propiedad del archivo de texto, o bien si nunca se ha ejecutado WriteProperty (por ejemplo si el valor no ha cambiado nunca respecto al valor predeterminado).

Agregar una página

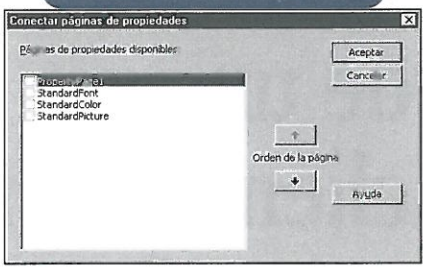
Las propiedades que hemos añadido en el apartado anterior, se mostrarán automáticamente en la ventana de propiedades de VB, pero también podemos observar éstas en unas fichas alternativas (como por ejemplo en el control SSTab, la ficha que aparece al pulsar el botón Personalizado). Utilizaremos dichas páginas de propiedades, cuando su uso simplifique el tratamiento de las mismas para el usuario, o cuando deseemos distribuir nuestro control internacionalmente y pretendamos que lo entienda todo el mundo.

Empezaremos en el menú Proyecto, donde se hará clic en Agregar páginas de propiedades. Cambiamos la propiedad Name por EjemploSPProp. Guardamos ahora la página de proyectos (Archivo + Guardar grupo de proyectos). VB le asignará automáticamente la extensión .pag. Si dicha página tuviese información de tipo binario (por ejemplo mapas de bits) se guardarían en un archivo con el mismo nombre pero con la extensión .pgx.

En dicha página colocamos una etiqueta y una caja de texto (con la propiedad Name igual a txtTitulo), hacemos doble clic en la página de propiedades para ir a la ventana de texto y en ella es donde elegimos el evento SelectionChanged, añadiendo el código que se indica a continuación:

```
Private Sub PropertyPage_SelectionChanged()
  txtTitulo.Text = SelectedControls(0).Caption
End Sub
```

Podemos conectar múltiples páginas de propiedades a un control.



Ponemos el código en este evento con el fin de obtener los valores existentes de las propiedades de control, ya que puede ser que exista más de un control seleccionado. Este evento se producirá cada vez que abramos la página de propiedades o cuando cambia la lista de controles seleccionados (puede cambiar ya que la página de propiedades no es modal y mientras está abierta puede ser que el usuario seleccione otros controles).

El evento ApplyChanges sirve para establecer los valores de las propiedades correspondientes a los controles seleccionados, por ejemplo:

```
Private Sub PropertyPage_ApplyChanges()
Dim obj As Variant
For Each obj In SelectedControls
  obj.Caption = txtTitulo.Text
Next
End Sub
```

Dicho evento se producirá cuando el usuario pulse el botón de Aceptar o Aplicar, de la página de propiedades.

Las páginas de propiedades normalmente tendrán los botones de Cancelar, Aceptar y Aplicar. El botón Aplicar, se activa cuando se produce algún cambio, para esto escribimos en el evento Change de la caja de texto:

```
Private Sub txtTitulo_Change()
  Changed = True
End Sub
```

Abrimos ahora el diseñador de nuestro control ActiveX, y en la ventana de propiedades hacemos clic en la propiedad PropertyPages, que nos permitirá conectar múltiples páginas de propiedades a un control de usuario y controlar el orden en que estas aparecerán.

Seleccionamos nuestra página y ahora cada vez que pulsemos el botón derecho sobre un control, y una vez allí escojamos la opción de propiedades, veremos la página de propiedades que hemos creado. Aunque el tema de las páginas de propiedades es un poco más amplio lo dejamos para un análisis posterior en el próximo artículo.

Agregar eventos

Antes de empezar este apartado debemos distinguir entre dos clases de eventos, por una parte los eventos que recibirá nuestro control y por otra parte los que provocará. Éstos originarán que el programador pueda escribir código en ellos para realizar las acciones que desee.

Aunque el tema de los eventos es bastante complejo, vamos a ver un ejemplo simple de cómo agregar un evento a nuestro control (evento clic o pulsación del ratón sobre él).

Para comenzar abrimos la página de Código del Control, y en la sección correspondiente a declaraciones, añadimos el siguiente código:

```
Public Event Click()
```

Y ahora añadimos el código siguiente de la etiqueta:

```
Private Sub LabelPrueba_Click()
  RaiseEvent Click
End Sub
```

Este trozo de código, sólo desencadenará el evento si hacemos un clic sobre la etiqueta, si quisiéramos que esto sucediese en todo el control, habríamos de añadir el mismo código, al fondo, y a la caja de texto.

PROGRAMACIÓN DE SOCKETS

Creación de un cliente Echo

Enrique de la Lastra (miluen@redestb.es)

Para programar una aplicación Internet completa es imprescindible dominar los Windows Sockets. Haremos uso de ellos para crear un objeto Socket y desarrollaremos un cliente echo simple, para probarlo.

Los Windows Sockets representan la implementación en código de los protocolos TCP/IP para el entorno Windows (Windows 3.11 y Windows 95). Se distribuyen en forma de librería DLL: "WINSOCK.DLL" (con 16 bits) o bien "WSOCK32.DLL" (32 bits), de manera que todas nuestras aplicaciones de Internet harán uso de la API de esa librería, olvidándose de aquello que pueda ocurrir debajo, en la red. Desde la perspectiva del programador de aplicaciones iNet (Internet/Intranet) basta conocer y entender las funciones definidas en la WINSOCK.DLL, para realizar cualquier aplicación TCP/IP.

La API de los Windows Sockets está basada en la implementada en la versión 4.3 del UNIX BSD (*Berkeley Software Distribution*). Su actualización es continua a través de las sugerencias que van llegando al foro de discusión electrónica (winsock@mycrolyne.com). Actualmente la última versión es la 2.0, aunque todavía no está soportada en la implementación *Winsock* de Microsoft. La versión 2.0 es totalmente compatible con la versiones anteriores y añade nuevas funcionalidades como, por ejemplo, el soporte de otros protocolos distintos de TCP/IP sobre la misma interfaz API, facilidades de resolución de nombres independientes del protocolo de red (por ejemplo: DNS, *Domain Name System*), aprovechamiento de las capacidades *multicast* (multidestino) de la capa de transporte en caso de que las tenga, y otras.

En las siguientes líneas, se mostrará la creación de un objeto *SOCKET* que será utilizado para desarrollar un software cliente del protocolo *echo* (eco). Éste nos permite comprobar el estado de una conexión TCP/IP. El mismo protocolo está definido en la "rfc862" (las RFC's *-Request For Comment- son* documentos en los que se especifican los protocolos de Internet). Lo veremos más adelante en este artículo.

Algunas aclaraciones previas

En la revista publicada tres números antes que ésta, comenzamos la creación de un objeto Delphi que denominamos: "*TSocket_sp*" y que era el germen de nuestra propia clase "*Windows socket*". A lo largo del artículo desarrollaremos el objeto completo, de modo que para aquellos que no disponen de esa revista, incluimos la clase *TSocket*_sp, que comenzamos en dicho artículo, en el Listado 1. También para ellos, recordamos la definición de *socket*: es la combinación entre una dirección IP y un número de puerto TCP. Ese número, o pareja DIRECCIÓN IP + NÚMERO DE PUERTO TCP, define una aplicación de red única en el mundo.

Como vimos en ese artículo, a partir de la versión 2.0 de Delphi, existe un fichero "WINSOCK.PAS", que redefine en código Delphi todas las funciones pertenecientes a la API de la librería "WINSOCK.DLL". La localización de este fichero es el directorio:

C:\Directorio de DELPHI 2.0 ó 3.0\Source\RTL\WIN

Para los que todavía programen con Delphi 1.0, se incluyen en el CD-ROM de la revista los archivos "winsock.pas" y "winsock.inc" que si bien utilizan en algún caso distintos nombres de tipos que los definidos en el fichero "winsock.pas" de Delphi 2.0 ó 3.0, se puede decir que son prácticamente idénticos.

Con la versión 2.0 se dispone de un fichero "WINSOCK.PAS", que redefine las funciones de la API pertenecientes a "WINSOCK.DLL"

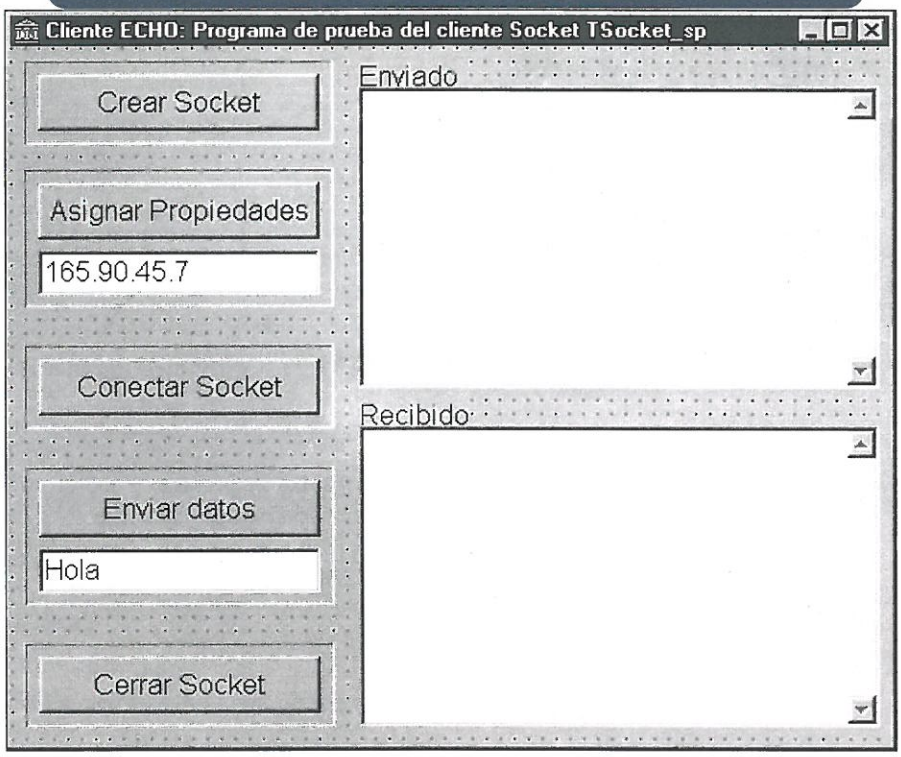
También se encuentran en el CD-ROM las especificaciones 1.1 y 2.0 de los *Windows Sockets*, en documentos de texto, hipertexto y formatos "write" y "word". El archivo al que haremos referencia cuando hablemos de la especificación de la API de los *Windows Sockets* será el documento de word "Wsock20_api.doc".

El objeto *socket* que vamos a desarrollar (*TSocket_sp*) hará uso de las funciones redefinidas en la *unit* "WINSOCK.PAS" y creará una interfaz fácil para las aplicaciones de red que quieran usar los servicios TCP/IP.

Unos incisos antes de continuar:

1. recordar que el tipo *TSocket* está definido en "WINSOCK.PAS" como *integer*. Cada vez que creemos un *socket* para una aplicación concreta, la función que lo crea devuelve un valor entero que representa dicho *socket* y que deberemos utilizar en todas las funciones *Winsock* de entrada y salida de las que haga uso la aplicación. A causa de esto, conviene siempre recordar que *TSocket* hace referencia al descriptor de un *socket* concreto, mientras que *TSocket_sp* hace referencia a la clase (objeto) que vamos a definir.

Figura 1: Presentación del módulo de prueba echo.



2. Como se ha venido explicando en el contenido de este texto, la versión 1.1 es la última soportada por la implementación Winsock de Microsoft. A pesar de todo, y siempre que nos sea posible, intentaremos basarnos en la versión 2.0 a lo largo del artículo. Sin embargo, si llamamos a *WSAStartup()* utilizando $0200 (2.0) como valor para el parámetro de versión, la función retornará el siguiente error: WSAVERNOTSUPPORTED (que significa versión *Winsock* no soportada).

3. En la función *CrearSocket()* (ver Listado 1) se llama a la función *WSAGetLastError()* en el caso de que al intentar crear el socket el resultado sea *INVALID_SOCKET* (socket no válido). Dado que la función *WSAGetLastError()* devuelve un entero con el valor del error, el mensaje que se muestra en pantalla no es nada explicativo del error. Para un mejor control de los errores, podemos definir una función que devuelva una cadena de texto aclarativa del error, en lugar de la constante que define el error. La explicación de estas constantes se detalla para cada función en la parte: "*Error Codes*" (códigos de error) dentro del documento "Wsock20_api.doc". En el Listado 2, se puede ver la función encargada de tratar los errores, que hemos denominado *ObtenerMensajeError()*.

4. Para dar mayor consistencia a la definición de las funciones, se intentará, siempre que sea posible, que éstas devuelvan un *boolean* que indique el éxito o fracaso de la operación.

Tabla 1: Eventos ante los cuales la función WSAAsyncSelect() enviará un mensaje a la ventana elegida.

EVENTO	SIGNIFICADO
FD_READ	Socket preparado para lectura
FD_WRITE	Socket preparado para escritura
FD_OOB	Datos fuera de banda preparados para lectura en el socket
FD_ACCEPT	Socket preparado para aceptar una nueva conexión entrante
FD_CONNECT	La conexión iniciada en el socket se ha completado
FD_CLOSE	La conexión del socket se ha cerrado
FD_QOS	La calidad del servicio del socket ha cambiado
FD_GROUP_QOS	La calidad del servicio del grupo de sockets al que pertenece el socket ha cambiado

También redefiniremos en la clase "TSocket_sp" aquéllos tipos y constantes de la interfaz *Winsock* que queramos usar en los servicios de red. De esta forma, aislamos al máximo lo que ocurre a un lado y al otro de los *Windows Sockets*. La función "CrearSocket()" la redefiniremos del siguiente modo:

```
function TSocket_sp.CrearSocket: sock;
begin
 SocketCreado := socket(PF_INET, SOCK_STREAM, 0);
 if (SocketCreado = INVALID_SOCKET) then
    begin
  Result := SOCKET_INVALIDO;
  ShowMessage('Error en CrearSocket: ' +
    IntToStr(WSAGetLastError));
 end
 else
  Result := SocketCreado;
end;
```

Como hemos visto, utilizamos una nueva variable: "SocketCreado". Dicha variable recogerá el valor del último *socket* que hayamos creado y situaremos en la parte *private* (con el fin de que no sea accesible desde fuera de la clase):

```
private
 SocketCreado: sock;
```

El tipo "sock" es una redefinición del tipo "TSocket" para que pueda ser utilizado por la aplicación que haga uso de la clase "TSocket_sp", sin necesidad de hacer referencia en la parte *uses* a la *unit "Winsock"*. De igual manera, está redefinida la constante "SOCKET_INVALIDO":

```
const
 SOCKET_INVALIDO = INVALID_SOCKET;

type
 sock = TSocket;
```

Si la llamada a la función *CrearSocket()* tiene éxito (resultado distinto de SOCKET_INVALIDO), la variable "SocketCreado" almacena el valor del *socket*. En caso contrario, se habrá producido un error y, por tanto, no interesa el valor de la variable "SocketCreado".

5. Para crear un componente Delphi en lugar de un objeto (y poder manipularlo en tiempo de diseño desde *Object Inspector*), redefinimos la clase:

```
TSocket_sp = class(TComponent)
```

y creamos el constructor y el destructor de la misma:

```
public
 constructor Create(AOwner: TComponent); override;
 destructor Destroy; override;
```

En la parte implementation se añade el siguiente código:

```
constructor TSocket_sp.Create(AOwner:
      TComponent);
begin
 inherited Create(AOwner);
end;

destructor TSocket_sp.Destroy;
begin
 inherited Destroy;
end;
```

Modo asíncrono y modo bloqueante

Tal y como se muestra el objeto *TSocket_sp* en el Listado 1, no tiene funcionalidad alguna, ya que lo único que puede hacerse con él es crearlo, abrirlo, cerrarlo y destruirlo. Vamos a conseguir que sirva para algo, pero antes de comenzar debemos conocer unos conceptos.

Los sockets pueden ser "bloqueantes" o "asíncronos". En el primer caso, el software de aplicación queda bloqueado ante cada llamada a funciones del socket

Debemos distinguir entre los dos modos de funcionamiento posibles de los *sockets*: bloqueante y asíncrono (existe un tercer modo, síncrono, al que no vamos a hacer referencia por el momento). En el modo asíncrono, el *software* de aplicación retoma el control después de utilizar un *socket* sin esperar la respuesta; en el modo bloqueante, en cambio, no se

devuelve el control al *software* de aplicación hasta que no se recibe la respuesta. En los sistemas operativos en los que existe multitarea real (Windows 95, Windows NT, UNIX...), al ocupar cada tarea su propio espacio de memoria independiente, el modo bloqueante no representa mayor problema ya que, salvo el programa que esté haciendo uso de los *Windows Sockets*, que se quedará paralizado hasta recibir una respuesta, el resto puede funcionar como lo hacía hasta ese momento. Sin embargo, en Windows 3.1 el modo bloqueante paraliza todo el sistema hasta que el *socket* recibe su respuesta. Esto lo soluciona el S.O. con un método que se encarga de gestionar los mensajes de Windows mientras llega la respuesta.

En general, será más interesante trabajar en modo asíncrono, pues es la forma más limpia y más eficiente de programar una aplicación. Es más limpia, porque tratamos "cada cosa en su sitio": la petición por una parte, y la respuesta en el manejador del evento que aquélla genere. Por otro lado, es más eficiente, porque no deja colgado el sistema en espera de una respuesta (que, por otro lado, puede no llegar).

Existen determinadas funciones *Winsock* asíncronas y funciones *Winsock* bloqueantes; de hecho, las más comunes son estas últimas. Esto en un principio podría parecer que está limitando las implementaciones asíncronas. Sin embargo, en la propia interfaz *Winsock* se está suministrando un mecanismo que sirve para "convertir" cualquier función bloqueante en asíncrona: la función *WSAAsyncSelect()*.

Suele ser más interesante trabajar con el modo asíncrono, porque la programación resulta más limpia y de mayor eficiencia

Tabla 2: Funciones de conversión de enteros desde el orden de almacenamiento de Intel al orden utilizado en Internet y viceversa.

FUNCIÓN	CONVERSIÓN DE BYTES	TIPOS DE ENTEROS
function htonl (hostlong : u_long) : u_long;	De Intel a Internet	Enteros largos (4 bytes)
function htons (hostshort : u_short) : u_short;	De Intel a Internet	Enteros normales (2 bytes)
function ntohl (netlong : u_long) : u_long;	De Internet a Intel	Enteros largos (4 bytes)
function ntohs (netshort : u_short) : u_short;	De Internet a Intel	Enteros normales (2 bytes)

Con el objetivo de activar el modo asíncrono, es necesario llamar a la función: *WSAAsyncSelect()*, a la que se deben pasar, entre otros parámetros, los eventos para los que el *socket* enviará un mensaje al programa. Al activar el modo asíncrono se devuelve el control al programa de aplicación, después de la utilización de cualquier función del API *Winsock*. Cada vez que la función termine de efectuar su labor, se disparará el evento que hayamos determinado al llamar a la función *WSAAsnycSelect()*. Veremos el funcionamiento de esta función en detalle en el siguiente punto.

Cabe reseñar aquí que todas las funciones de la interfaz *Winsock* que comienzan con las letras *WSA (WinSock API)*, son traducciones de las funciones correspondientes de la implementación original de los *sockets*: la de BSD (*Berkeley Software Distribution*). A su vez, las funciones que comienzan con las letras *WSAAsync* son funciones asíncronas.

Socket cliente y socket servidor

El modelo cliente-servidor es aquél que emplean las aplicaciones de Internet para comunicarse y enviarse información. En este modelo, una aplicación cliente (o simplemente un "cliente") solicita un determinado servicio (por ejemplo información) a otra aplicación cuya misión es proporcionárselo. Esta última aplicación es un "*software* servidor" o simplemente un "servidor" que, como define su propio nombre, proporciona un servicio. Es decir, el cliente pide y el servidor da, y así transcurre el diálogo básico de cualquier aplicación de Internet (desde FTP hasta HTTP).

Por extensión se suele denominar "cliente" al ordenador que corre un programa cliente, mientras que se denomina "servidor" a la máquina que corre el programa servidor. Una máquina puede ser cliente para una aplicación y servidora para cualquier otra (por ejemplo un servidor *Web* que corra un cliente *telnet*) e incluso ser cliente y servidora de la misma aplicación (por ejemplo un servidor *Web* donde tengamos instalado un navegador para conectarnos a otros *Webs*).

De acuerdo con este modelo descrito, para tener la posibilidad de comunicarnos con una máquina remota a través de la red Internet, necesitamos disponer de una aplicación cliente. Por otro lado es necesario que en el otro extremo exista una aplicación que nos atienda, es decir, un servidor. Es imposible comunicarse con un sistema que no tenga una aplicación que esté "a la escucha" de peticiones. Por ejemplo, si queremos conectarnos al *Web* de Telefónica (con la dirección www.telefonica.es) necesitamos un programa cliente *Web* (un navegador como Netscape, por ejemplo) pero, también es requisito indispensable que exista un servidor *Web* en Telefónica que nos atienda.

Los *sockets* trabajan de la misma manera: podemos crear un *socket* cliente y un *socket* servidor; o también, un *socket* que sea a la vez cliente y servidor.

Listado 1: Código del objeto TSocket_sp, tal cual quedó en el anterior artículo.

```
unit Sock_sp_anterior;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Winsock;

type
TSocket_sp = class (TObject)
private
  DatosWinsock: TWSAData;
public
  function IniciarWinsock: boolean;
  function CrearSocket: TSocket;
  function CerrarSocket (s: TSocket): boolean;
  function TerminarWinsock: boolean;
end; {class TSocket_sp}

implementation

{ Iniciamos los servicios de WINSOCK.DLL, versión 1.1}
function TSocket_sp.IniciarWinsock: boolean;
begin
  if (WSAStartup($0101, DatosWinsock) = 0) then
    Result := True
  else
    Result := False;
end;

{ Creamos un Socket TCP/IP (PF_INET) orientado a conexión (SOCK_STREAM) }
function TSocket_sp.CrearSocket: TSocket;
begin
  Result := socket(PF_INET, SOCK_STREAM, 0);
  if (Result = INVALID_SOCKET) then
    ShowMessage('Error: ' + IntToStr(WSAGetLastError));
end;

{ Cerramos el Socket cuyo descriptor pasemos como parámetro }
function TSocket_sp.CerrarSocket(s: TSocket): boolean;
begin
  if closesocket(s) = 0 then
    Result := True
  else
    Result := False;
end;

{ Terminamos los servicios de WINSOCK.DLL }
function TSocket_sp.TerminarWinsock: boolean;
begin
  if WSACancelBlockingCall = 0 then
    Result := True
  else
    Result := False;
  if (WSACleanup <> 0) then
    Result := False;
end;

end.
```

Conectando nuestro socket a otro sistema

Vamos a añadir a nuestro componente las funciones necesarias para conectarlo con un *socket* de otro ordenador de Internet. Esto es, vamos a definir las funciones que convertirán al componente en un *socket cliente*.

Conectar un socket a otro sistema consiste en convertir dicho elemento en cliente

La primera de estas funciones será la que permita abrir la conexión con otro sistema. En el API *Winsock* se definirá para este objetivo la función: *connect()*, de la siguiente forma:

```
function connect (s: TSocket; var name:
      TSockAddr; namelen: integer): integer
```

La función devuelve un valor "cero" si la conexión tuvo éxito y el código del error SOCKET_ERROR en caso contrario. Para estudiar y tratar los códigos de error devueltos -por la función *WSAAsyncSelect()*- habrá que dirigirse al documento de la especificación *Winsock*. De estos códigos de error, los que nos interesan están tratados en la función *ObtenerMensajeError()* que se puede observar en el Listado 2.

De los tres parámetros que hay que pasar a *connect()*, el primero es el descriptor del socket, (un valor entero) que devuelve la función *"socket()"* al crear uno. A continuación lo almacenamos en la variable *"SocketCreado"*. El segundo parámetro, de tipo "TSockAddr", define la dirección Internet (IP) del ordenador al que vamos a conectarnos y el número de puerto (TCP) de la aplicación de nivel

superior. El tercer parámetro que se describe es el tamaño del registro "TSockAddr" por lo que basta pasar el valor: "SizeOf(TSockAddr)".

El primer paso que permita abrir la conexión con otro sistema es definir la función connect en el API Winsock

El tipo "TSockAddr" es una redefinición literal del tipo "TSockAddrIn", que, según podemos comprobar en el fichero "WINSOCK.PAS", se define como se muestra a continuación:

```
TSockAddrIn = packed record
  case Integer of
    0: (sin_family: u_short;
        sin_port: u_short;
        sin_addr: TInAddr;
        sin_zero: array[0..7] of Char);
    1: (sa_family: u_short;
        sa_data: array[0..13] of Char)
end;
```

Al ser un registro "empaquetado" (*packed record*) las partes situadas a ambos "lados" de la sentencia *"case"* comparten memoria, por lo que se puede llenar uno de los "lados" y hacer uso de las variables del otro. En nuestro caso, sólo utilizaremos las variables del *"case 0"*. Nos disponemos a analizar una a una dichas variables:

1. La variable "sin_family" almacena el tipo de dirección; como en nuestro caso será una "dirección IP", le asignaremos la constante: "PF_INET".
2. La variable "sin_port" contiene el número del puerto TCP que utilizará el *SocketCreado*.
3. La variable "sin_addr" almacena el valor de la dirección IP. El tipo de esta variable se define:

Listado 2: Código del objeto TSocket_sp finalizado.

```
unit sock_sp;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, Winsock;

const
SOCKET_INVALIDO   = INVALID_SOCKET;

CONECTADO        = FD_CONNECT;
ENVIO_FINALIZADO  = FD_WRITE;
DATOS_RECIBIDOS   = FD_READ;

WMS_MENSAJESOCKET = WM_USER + 1;

type
sock = TSocket;

type
TSocket_sp = class (TComponent)
private
  DatosWinsock: TWSAData;
  DirSocketDestino: TSockAddr;
  bufferRx: PChar;
  MensajeError: string;
  function ObtenerDirInternet (LaDireccion: string): longint;
  procedure ObtenerMensajeError (funcion: string);
private
  FDirInternet: string;
  FPuertoTCP: integer;
  procedure SetDirInternet (AValue: string);
  procedure SetPuertoTCP (AValue: integer);
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
public
  function IniciarSocket: boolean;
  function CrearSocket: sock;
  function ConectarSocket (s: sock): boolean;
  function CerrarSocket  (s: sock): boolean;
  function EnviarEnSocket (s: sock; cadena: string): boolean;
  function RecibirEnSocket(s: sock; var cadena: string): boolean;
  function TerminarSocket: boolean;
published
  property DirInternet: string read FDirInternet write SetDirInternet;
  property PuertoTCP: integer read FPuertoTCP write SetPuertoTCP;
end; {class TSocket_sp}

implementation

constructor TSocket_sp.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FPuertoTCP:= 7; { puerto TCP para echo};
  FDirInternet:= '127.0.0.1'; { host local }
  DirSocketDestino.sin_family := PF_INET;
  DirSocketDestino.sin_zero := char(0);
end;
```

Listado 2. Continuación

```
destructor TSocket_sp.Destroy;
begin
  inherited Destroy;
end;

{ Iniciamos los servicios de WINSOCK.DLL, versión 1.1}
function TSocket_sp.IniciarSocket: boolean;
begin
  Result := False;
  case WSAStartup($0101, DatosWinsock) of
     0              : Result := True;
     { Si el valor devuelto no es cero, recogemos el mensaje del error
       que se ha producido }
     WSASYSNOTREADY    : MensajeError:= 'Red no preparada para la comunicación';
     WSAVERNOTSUPPORTED: MensajeError:= 'Versión 1.1 no soportada por la implementación WINSOCK';
     WSAEINPROGRESS    : MensajeError:= 'Una operación Winsock bloqueante está en progreso';
     WSAEPROCLIM       : MensajeError:= 'El límite de tareas Winsock ha sido sobrepasado';
     else              MensajeError:= '(no registrado) ' + IntToStr(WSAGetLastError);
  end;
  if not Result then begin
    MensajeError := 'Función: "IniciarSocket"' + #13 + #13 + 'ERROR: ' + MensajeError;
    ShowMessage(MensajeError);
  end;
end;

{ Creamos un Socket TCP/IP (PF_INET) orientado a conexión (SOCK_STREAM) }
function TSocket_sp.CrearSocket: sock;
begin
  Result := socket(PF_INET, SOCK_STREAM, 0);
  if (Result = INVALID_SOCKET) then
    ObtenerMensajeError ('CrearSocket');
end;

procedure TSocket_sp.SetDirInternet (AValue: string);
var lDirInternet: longint;
begin
  lDirInternet := ObtenerDirInternet (AValue);
  { Comprobamos que se ha introducido una dirección Internet válida }
  if lDirInternet <> INADDR_NONE then begin
    FDirInternet:= AValue;
    DirSocketDestino.sin_addr.s_addr := lDirInternet;
  end;
  DirSocketDestino.sin_port := htons(FPuertoTCP);
end;

procedure TSocket_sp.SetPuertoTCP (AValue: integer);
begin
  { Primero hay que fijar la Dirección Internet.
    Segundo el puerto TCP.
    Tercero volver a fijar la Dirección Internet }
  FPuertoTCP := AValue;
  { Almacenamos el valor del puerto en el formato de Internet,
    en el que el orden de los bytes es: primero el byte mayor y al final
    el byte de menor valor }
  DirSocketDestino.sin_port := htons(FPuertoTCP);
  SetDirInternet (FDirInternet);
end;
```

```
TInAddr = packed record
  case integer of
    0: (S_un_b: SunB);
    1: (S_un_w: SunW);
    2: (S_addr: u_long);
end;
```

De la misma manera que antes, se trata de un registro empaquetado, del que sólo utilizaremos la variable: "S_addr". Esta variable sirve para definir la dirección IP. Como aquí esta dirección se almacena como un entero largo y, normalmente trabajamos con direcciones IP en el formato de cadena de caracteres (es decir: "123.123.123.123"), podemos hacer uso de la función "inet_addr" que realiza la correspondiente conversión. Esta función se declara en el fichero "WINSOCK.PAS":

```
function inet_addr(cp: PChar): u_long;
```

4. La variable "sin_zero" debe ir rellena de caracteres "cero" (#0).

   Al igual que con el valor del último *socket* creado, crearemos en la clase "TSocket_sp" una variable interna y privada del tipo "TSockAddr" que rellenaremos con los datos necesarios para conectarse a otro sistema:

```
private
  DirSocketDestino: TSockAddr;
```

En el método *Create* inicializamos los valores fijos de esta variable:

```
constructor TSocket_sp.Create(AOwner:
      TComponent);
begin
  inherited Create(AOwner);
  DirSocketDestino.sin_family := PF_INET;
  DirSocketDestino.sin_zero := #0;
end;
```

Como la función *connect()* es bloqueante, en cuanto tengamos todos los parámetros dispuestos para pasárselos a esta función, debemos decidir si queremos que se comporte de forma asíncrona. En caso afirmativo, llamaremos previamente

a la función *WSAAsyncSelect()*, que está declarada de la manera que se muestra en las líneas siguientes:

```
function WSAAsyncSelect (s: TSocket; Hwindow:
    HWND; wMsg: u_int; lEvent: longint): inte-
    ger;
```

Como siempre, el valor devuelto será cero si todo ha ido bien y, en caso contrario, recogerá el código del error.

Para indicar al Winsock que nos envíe mensajes relacionados con los eventos de red, se realiza una llamada a WSAAsyncSelect()

El primer parámetro es el descriptor del *socket* (SocketCreado). El segundo, el *handle* de la ventana que recibirá los mensajes pertinentes (en nuestro caso, fijaremos automáticamente el valor del *handle* al del padre de nuestro componente). El tercer parámetro es un mensaje, definido por nosotros que será el que habrá enviado la función *WSAAsyncSelect()* a la ventana que se haya elegido, cuando ocurra alguno de los eventos que hayamos determinado en el cuarto parámetro.

Los eventos de red ante los cuales la función *WSAAsyncSelect()* enviará un mensaje a la ventana elegida están resumidos en la Tabla 1. Nosotros debemos elegir entre ellos los que queramos activar, pasándoselos a *WSAAsyncSelect()*. Con la finalidad de activar varios de estos bastará con realizar un "OR" de los mismos (por ejemplo, para activar "FD_CONNECT" y "FD_READ" emplearemos "(FD_CONNECT or FD_READ)" como valor del parámetro "lEvent"). De todos estos eventos, por ahora únicamente nos interesa el evento "FD_CONNECT", que informa que la función *connect()* ha terminado.

Listado 2. Continuación

```
function TSocket_sp.ObtenerDirInternet (LaDireccion: string): longint;
var pDirInternet: PChar;
begin
  { Almacenamos el valor de la dirección, para lo cual hay que convertirla
    primero en un "PChar" y luego en un "longint" }
  GetMem (pDirInternet, length(FDirInternet) + 1);
  StrPCopy (pDirInternet, FDirInternet);
  Result := inet_addr(pDirInternet);
  FreeMem (pDirInternet);
end;

function TSocket_sp.ConectarSocket (s: sock): boolean;
begin
  WSAAsyncSelect (s, (Owner as TWinControl).Handle, WMS_MENSAJESOCKET,
FD_CONNECT);
  Result := True;
  if connect (s, DirSocketDestino, SizeOf(TSockAddr)) = SOCKET_ERROR then begin
    if WSAGetLastError <> WSAEWOULDBLOCK then begin
      Result := False;
      ObtenerMensajeError ('ConectarSocket');
    end;
  end;
end;

function TSocket_sp.EnviarEnSocket (s: sock; cadena: string): boolean;
var bufferTx: PChar;
begin
  Result := True;
  WSAAsyncSelect (s, (Owner as TWinControl).Handle, WMS_MENSAJESOCKET,
FD_READ or FD_WRITE);
  cadena := cadena + #13#10;

  GetMem (bufferTx, Length(cadena) + 1);
  StrPCopy (bufferTx, cadena);
  if send (s, bufferTx, Length(cadena), 0) = SOCKET_ERROR then begin
    if WSAGetLastError <> WSAEWOULDBLOCK then begin
      Result := False;
      ObtenerMensajeError ('EnviarEnSocket');
      FreeMem (bufferTx);
    end;
  end;
end;

function TSocket_sp.RecibirEnSocket (s: sock; var cadena: string): boolean;
begin
  Result := True;
  GetMem (bufferRx, 1000);
  if recv (s, bufferRx, 256, 0) = SOCKET_ERROR then begin
    if WSAGetLastError <> WSAEWOULDBLOCK then begin
      Result := False;
      ObtenerMensajeError ('RecibirEnSocket');
      cadena := '';
      FreeMem (bufferRx);
    end
  end
  else begin
    cadena := StrPas (bufferRx);
    cadena := Copy (cadena, 1, Length(cadena)-2);
  end;
end;
```

Listado 2. Continuación

```
{ Cerramos el Socket cuyo descriptor demos como parámetro }
function TSocket_sp.CerrarSocket(s: sock): boolean;
begin
 Result := True;
 if closesocket(s) <> 0 then begin
  Result := False;
  ObtenerMensajeError ('CerrarSocket');
 end;
end;

{ Terminamos los servicios de WINSOCK.DLL }
function TSocket_sp.TerminarSocket: boolean;
begin
 Result := True;
 if WSACancelBlockingCall <> 0 then
  Result := False;
 if (WSACleanup <> 0) then
  Result := False;
 if not Result then
  ObtenerMensajeError ('TerminarSocket');
end;

procedure TSocket_sp.ObtenerMensajeError (funcion: string);
begin
 case WSAGetLastError of
  { WSAStartup() }
  WSANOTINITIALISED : MensajeError:= 'No se ha llamadao a WSAStartup()';
  WSAENETDOWN       : MensajeError:= 'La red o el Proveedor de servicios ha
fallado';
  WSAEAFNOSUPPORT   : MensajeError:= 'La familia de direcciones no está soportada';
  WSAEINPROGRESS    : MensajeError:= 'Una llamada WinSock 1.1 bloqueante está en
progreso';
  WSAEMFILE         : MensajeError:= 'Se han terminado los sockets disponibles';
  WSAENOBUFS        : MensajeError:= 'No hay memoria para crear un nuevo socket';
  WSAEPROTONOSUPPORT: MensajeError:= 'El protocolo especificado no está soporta-
do';
  WSAEPROTOTYPE     : MensajeError:= 'El protocolo especificado no concuerda con el
socket';
  WSAESOCKTNOSUPPORT: MensajeError:= 'El socket no está soportado en esta familia
de direcciones';
  { errores adicionales para: connect() }
  WSAEADDRINUSE     : MensajeError:= 'La dirección especificada ya está en uso';
  WSAEALREADY       : MensajeError:= 'Una llamada no bloqueante a connect() está en
progreso en el socket';
  WSAEADDRNOTAVAIL  : MensajeError:= 'La dirección no está disponible en la máquina
local';
  WSAECONNREFUSED   : MensajeError:= 'Connexión rechazada';
  WSAEFAULT         : MensajeError:= 'Parámetros no válidos';
  WSAEINVAL         : MensajeError:= 'El socket es de escucha';
  WSAEISCONN        : MensajeError:= 'El socket ya ha sido conectado';
  WSAENETUNREACH    : MensajeError:= 'Red inalcanzable desde esta máquina en este
momento';
  WSAENOTSOCK       : MensajeError:= 'El descriptor no es un socket';
  WSAETIMEDOUT      : MensajeError:= 'Conexión ha caído por fallo en la red o por falta
de respuesta';
  WSAEWOULDBLOCK    : MensajeError:= 'La conexión no puede completarse inmedia-
tamete';
  WSAEACCES         : MensajeError:= 'Intento de broadcast fallido';
  { errores adicionales para: send() }
  WSAENETRESET      : MensajeError:= 'La conexión debe reiniciarse porque Winsock la
ha tirado';
  WSAENOTCONN       : MensajeError:= 'El socket no está conectado';
```

El mensaje Windows que nos enviará la función *WSAAsyncSelect()* será definido, por ejemplo, de la forma que se va a mostrar con este código:

```
const
 WMS_MENSAJESOCKET = WM_USER + 1;
```

Los parámetros que habrá que pasarle a esta función serán:

1. s -> SocketCreado
2. Hwindow -> (Owner as TWinControl).Handle
3. wMsg -> *WMS_MENSAJESOCKET*
4. lEvent -> FD_CONNECT

De acuerdo con ellos se enviará el mensaje *WMU_MENSAJESOCKET* a la ventana padre cuando la función *connect()* haya finalizado.

Cada llamada que se haga a la función *WSAAsyncSelect()* provoca una anulación de la anterior llamada a la misma. Esto nos permite cambiar los eventos de red ante los que queremos que nos avise, o cambiar la ventana a la que queremos enviar los mensajes. Con el fin de cancelar las notificaciones, es decir, para indicar al *Winsock* que no envíe más mensajes relacionados con eventos de red, realizaremos una llamada a *WSAAsyncSelect()* con los dos últimos parámetros puestos a "cero":

```
WSAAsyncSelect (SocketCreado, hWnd, 0, 0);
```

En este punto ya estamos en condiciones de definir nuestra función de apertura de conexión del *socket* con otro sistema remoto. La estructura de la misma será como se muestra:

```
function TSocket_sp.ConectarSocket (s: sock):
      boolean;
begin
 WSAAsyncSelect (s, (Owner as
      TWinControl).Handle, WMS_MENSAJE-
      SOCKET, FD_CONNECT);
 Result := True;
 if connect (s, DirSocketDestino,
      SizeOf(TSockAddr)) = SOCKET_ERROR
      then begin  if WSAGetLastError <> WSA-
```

```
    EWOULDBLOCK then begin    Result :=
    False;
   ObtenerMensajeError ('ConectarSocket');
  end;
 end; end;
```

Se comprueba si hubo error y, en el caso de que fuese *WSAEWOULDBLOCK*, sabremos que todo fue bien. La razón es que este error sólo indica que estamos trabajando en modo asíncrono (no bloqueante), que es lo que queremos.

Definición de propiedades útiles

A partir de este momento se van a definir dos propiedades, que resultarán muy prácticas, en el componente *TSocket_sp*: la dirección Internet destino y el puerto TCP, de forma que estos valores sean modificables desde el *Object Inspector* de Delphi. A su vez, una vez han sido introducidos por el usuario los valores de estas propiedades, sus métodos inicializarán las variables internas que sean consideradas necesarias.

```
private
 FDirInternet: string;
 FPuertoTCP: integer;
 procedure SetDirInternet (AValue: string);
 procedure SetPuertoTCP (AValue: integer);
published
 property DirInternet: string read FDirInternet
      write SetDirInternet;
 property PuertoTCP: integer read FPuertoTCP
      write SetPuertoTCP;
```

En el método Create habrá que inicializar estas variables:

```
constructor TSocket_sp.Create(AOwner:
      TComponent);
begin
 inherited Create(AOwner);
 FDirInternet:= '127.0.0.1'; { host local }
 FPuertoTCP:= 0; { puerto 0};
 DirSocketDestino.sin_family := PF_INET;
 DirSocketDestino.sin_zero := char(0);
end;
```

Listado 2. Continuación

```
   WSAEOPNOTSUPP    : MensajeError:= 'MSG_OOB fue especificado pero el socket no es
SOCK_STREAM';
   WSAESHUTDOWN     : MensajeError:= 'El socket ha sido eliminado';
   WSAEMSGSIZE      : MensajeError:= 'El socket es SOCK_DGRAM, y el datagrama es
mayor que el máximo';
   WSAECONNABORTED  : MensajeError:= 'El cicuito virtual ha caído (Time-out?)';
   WSAECONNRESET    : MensajeError:= 'El cicuito virtual ha sido reiniciado en el otro
extremo';
   { errores adicionales para: recv() }
   { errores adicionales para: closesocket() }
   { errores adicionales para: WSACleanup() }
   { errores adicionales para: WSACancelBlockingCall() }
   else
    MensajeError:= '(no registrado) ' + IntToStr(WSAGetLastError);
  end;
 MensajeError := 'Función: "' + funcion + '"' + #13 + #13 + 'ERROR: ' + MensajeError;
 ShowMessage(MensajeError);
end;

end.
```

Existen funciones en la interfaz WINSOCK que permiten cambiar el orden de almacenamiento desde la nomenclatura de Intel a la de Internet y viceversa

En cuanto a los métodos "SetDirInternet" y "SetPuertoTCP", deberán asignar los valores de la dirección IP y del puerto TCP a las variables correspondientes pertenecientes al registro "DirSocketDestino". Antes de detallar el cuerpo de estos métodos, es necesario introducir una aclaración referente al orden de almacenamiento de los bytes. En las máquinas Intel los bytes se almacenan comenzando por el byte de menor valor y terminando por el byte de orden mayor. Sin embargo, el orden de almacenamiento de los bytes es exactamente el contrario en las recomendaciones de Internet: primero, el byte de mayor orden y, después, el de menor orden. Esto quiere decir que antes de "escribir" un valor entero en Internet, debemos cambiar el orden de sus bytes. Y resulta muy fácil, pues la propia interfaz *Winsock* nos proporciona las funciones necesarias para realizarlo. Estas funciones se explican en la Tabla 2.

Aclarado este extremo, los métodos expuestos anteriormente presentan la estructura que se enumera en el código expuesto a continuación:

```
procedure TSocket_sp.SetDirInternet (AValue:
      string);
begin
 { Almacenamos el valor de la dirección, para lo
      cual hay que convertirla primero en un
      "PChar" y luego en un "longint"}
 lDirInternet := ObtenerDirInternet (AValue);

 { Comprobamos que se ha introducido una direc-
      ción Internet válida }
 if lDirInternet <> INADDR_NONE then begin
  FDirInternet:= AValue;
  DirSocketDestino.sin_addr.s_addr :=
      lDirInternet;
 end;
end;

procedure TSocket_sp.SetPuertoTCP (AValue:
      integer);
begin
 FPuertoTCP := AValue;
 { Almacenamos el valor del puerto en el formato
      de Internet, en el que el orden de los bytes
```

Listado 3. Código del programa de prueba echo.

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Sock_sp;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Bevel5: TBevel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Memo1: TMemo;
    Memo2: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    procedure WMSMensajeSocket(var Msg: TMessage); message WMS_MENSAJESOCKET;
  public
    SockEcho: sock;
    MiSocket_sp: TSocket_sp;
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  MiSocket_sp := TSocket_sp.Create (Self);
  MiSocket_sp.IniciarSocket;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  SockEcho := MiSocket_sp.CrearSocket;
  Memo1.Lines.Add ('Socket creado: ' + IntToStr(SockEcho));
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
    es: primero el byte mayor y al final el byte
    de menor valor }
  DirSocketDestino.sin_port :=
    htons(FPuertoTCP);
end;
```

La función "ObtenerDirInternet()" devuelve la dirección con tipo *longint* o el código de error "INADDR_NONE" en caso de que la dirección sea incorrecta:

```
procedure TSocket_sp.ObtenerDirInternet
    (LaDireccion: string): longint;
var pDirInternet: PChar;
begin
  GetMem (pDirInternet, length(FDirInternet) +
    1);
  StrPCopy (pDirInternet, FDirInternet);
  Result := inet_addr(pDirInternet);
  FreeMem (pDirInternet);
end;
```

Enviando información a través del socket

Para enviar datos a la red a través de un *socket* existe en la interfaz *Winsock* la función denominada *send():*

```
function send (s: TSocket; buf: PChar; len, flags:
    integer): integer;
```

Esta función retorna un valor entero que contiene el número de bytes enviados o el código de error SOCKET_ERROR. En este último caso expuesto, trataremos el código de error en la función *ObtenerMensajeError()* (Listado 2). Los parámetros de la función *send()* incluyen el descriptor del *socket*, el *buffer* con los datos a enviar, el tamaño del *buffer* y unos "*flags*" (indicadores) concretos.

Los "*flags*" se utilizan para cambiar el comportamiento de la función y pueden ser cualesquiera de los dos siguientes (o un "OR" de ellos): MSG_DONTROUTE,

para datos que no deben ser encaminados y, MSG_OOB, para datos fuera de banda (ver especificación para más detalles). Su uso queda fuera del alcance de nuestro pequeño *socket* y, por tanto, fijaremos el valor de este parámetro a cero.

Hay que tener en cuenta que al querer que esta función se comporte de forma asíncrona, debemos llamar de nuevo a *WSAAsyncSelect()* pero, esta vez, con los nuevos eventos con los que pretendemos que nos avise. Estos eventos serán: "FD_READ" (que informa de que hay datos esperando para ser recibidos) y "FD_WRITE" (que informa que la función *send()* ha terminado). Cabe recordar aquí que al efectuar una nueva llamada a la función *WSAAsyncSelect()* anulamos la anterior.

Nuestra función para enviar datos quedará de la siguiente forma:

```
function TSocket_sp.EnviarEnSocket (s: sock;
      cadena: string): boolean;
var bufferTx: PChar;
begin
 Result := True;
 WSAAsyncSelect (s, (Owner as
      TWinControl).Handle, WMS_MENSAJE-
      SOCKET, FD_READ or FD_WRITE);
 cadena := cadena + #13#10;

 GetMem (bufferTx, Length(cadena) + 1);
 StrPCopy (bufferTx, cadena);
 if send (s, bufferTx, Length(cadena), 0) = SOC-
      KET_ERROR then begin
  if WSAGetLastError <> WSAEWOULD-
      BLOCK then begin
   Result := False;
   ObtenerMensajeError ('EnviarEnSocket');
   FreeMem (bufferTx);
  end;
 end; end;
```

La cadena que se pasa como parámetro a la función "EnviarEnSocket()" la terminamos con un retorno de carro, más un salto de línea (caracteres #13 y #10), pues de esta manera terminan los comandos de las aplicaciones pertenecientes a Internet. En cuanto al *buffer* de transmisión, lo creamos con los datos pasados en el parámetro *cadena*. Este aspecto se puede observar en el código que se muestra anteriormente.

Listado 3. Continuación.

```
 MiSocket_sp.DirInternet := Edit1.Text;
 MiSocket_sp.PuertoTCP  := 7; { puerto echo }
 Memo1.Lines.Add ('Dirección y puerto asignados');
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
 MiSocket_sp.ConectarSocket (SockEcho);
 Memo1.Lines.Add ('Conectando con ' + Edit1.Text);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
 MiSocket_sp.EnviarEnSocket (SockEcho, Edit2.Text);
 Memo1.Lines.Add (Edit2.Text);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
 MiSocket_sp.CerrarSocket (SockEcho);
end;

procedure TForm1.WMSMensajeSocket(var Msg: TMessage);
var DatosRecibidos: string;
begin
  case Msg.LParamLo of
   CONECTADO:        Memo1.Lines.Add ('!CONECTADO!. Envíe datos');
   ENVIO_FINALIZADO:  Memo1.Lines.Add ('Envío finalizado. Envíe nuevos datos');
   DATOS_RECIBIDOS:   begin
                MiSocket_sp.RecibirEnSocket (SockEcho, DatosRecibidos);
                Memo2.Lines.Add (DatosRecibidos);
               end;
  end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 MiSocket_sp.TerminarSocket;
 MiSocket_sp.Free;
end;

end.
```

Recibiendo información con el socket

De la misma manera que existe una función para enviar datos en la interfaz *Winsock*, también es posible encontrar la función correspondiente que nos sirva para recibir datos:

```
function recv (s: TSocket; var buf; len, flags: inte-
      ger): integer;
```

Esta función es prácticamente idéntica a la anterior, con la diferencia de que ahora el parámetro "buf" debe ser un *buffer* de datos vacío, puesto que queremos recibir datos en él. Aquí no hará falta realizar una nueva llamada a *WSAAsyncSelect()*, debido a que los eventos son los mismos que los definidos en la función *EnviarEnSocket()*. Aquí

el evento que interesa es FD_READ, que proporciona la información necesaria sobre la existencia de datos disponibles para ser leídos.

Nuestra función *RecibirEnSocket()* es similar a la anterior, *EnviarEnSocket()*. La diferencia radica en que el parámetro *cadena* es "variable" de manera que tenga la posibilidad de recoger el contenido del *buffer* de recepción.

```
function TSocket_sp.RecibirEnSocket (s: sock; var
      cadena: string): boolean;
begin
 Result := True;
 GetMem (bufferRx, 1000);
 if recv (s, bufferRx, 256, 0) = SOCKET_ERROR
      then begin
  if WSAGetLastError <> WSAEWOULD-
      BLOCK then begin
   Result := False;
   ObtenerMensajeError ('RecibirEnSocket');
   cadena := '';
   FreeMem (bufferRx);
  end
 end
 else begin
  cadena := StrPas (bufferRx);
  cadena := Copy (cadena, 1, Length(cadena)-2);
 end;
end;
```

El resultado de todo esto es que finalmente hemos obtenido nuestro componente "TSocket_sp" ya terminado. El código completo se puede observar en el Listado 2 de este artículo. Con él podremos enviar y recibir datos de cualquier máquina donde exista un *socket servidor* que se esté ejecutando.

Es bastante claro que necesitaremos una aplicación que haga uso de el componente que se ha ido creando, con el objetivo de poder enviar y recibir los datos.

La aplicación que se requiere en este caso, puede ser un *software* realizado por nosotros mismos, que lleve a cabo las funciones personales. También es posible emplear una aplicación estándar de Internet: un cliente *FTP*, un cliente *telnet*, un cliente *HTTP*, un cliente de correo electrónico, etc, entre otras.

Cliente echo de prueba

Terminado el cuerpo de nuestro componente *socket cliente* básico, debemos comprobar que funciona. Realizaremos un programa de prueba que consistirá en un, cliente del protocolo *echo*, que permite comprobar que una conexión TCP/IP está funcionando (justo lo que queremos).

En este protocolo, definido en la "rfc862", el servidor está a la escucha de conexiones en el puerto TCP número 7. Una vez que se establece una conexión, el servidor devuelve exactamente los datos que ha recibido de dicha conexión, repitiendo el proceso hasta que el cliente ha logrado terminarla.

El protocolo echo permite comprobar que una conexión TCP/IP está funcionando. Nos proporciona la facilidad de probar que nuestro socket cliente funciona correctamente

El protocolo *echo* es una extensión del protocolo *ping*. En el protocolo *ping* el servidor tan sólo responde diciendo que "está vivo". Se llama *ping* porque funciona como el *ping-pong*: el cliente manda un *ping* y el servidor responde con otro *ping*. De esta manera es posible comprobar si el servidor está funcionando (si está encendido) o no (si está apagado).

El problema es que el protocolo *ping* no utiliza para su transmisión el protocolo TCP, sino sólo la parte IP. Esto significa que sólo comprueba que el protocolo IP funciona pero no dice nada acerca del buen funcionamiento del TCP.

Nuestro cliente *echo* será muy sencillo: únicamente tendrá que crear un *socket*, conectarse a través de él con el ordenador remoto (servidor) que queramos probar, escuchar las respuestas del servidor y cerrar el *socket* al final. El programa de prueba se muestra en el Listado 3, y en la Figura 1.

De nuestro cliente *echo*, merece destacar la función que maneja el mensaje WMS_MENSAJESOCKET. Esta función se declara en la parte *private* de la clase:

```
private
 procedure WMSMensajeSocket(var Msg:
      TMessage); message WMS_MENSAJESOC-
      KET;
```

Del parámetro *Msg* nos interesa la variable "lParamLo", que es la que almacena el código del evento. Como los códigos de eventos: FD_CONNECT, FD_READ y FD_WRITE, son internos al fichero "WINSOCK.PAS" los hemos redefinido en la clase "TSocket_sp" de la siguiente manera:

```
const CONECTADO       = FD_CONNECT;
      ENVIO_FINALIZADO = FD_WRITE;
      DATOS_RECIBIDOS  = FD_READ;
```

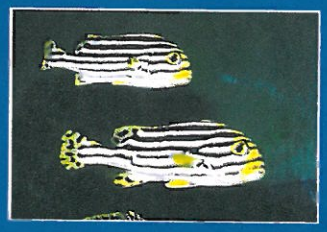
Conclusión

Aquí termina la introducción —que se inició hace tres números— a la programación de *sockets*. Haciendo uso de los conocimientos adquiridos y curioseando un poco en la especificación de la librería WINSOCK, es posible llegar a personalizar y perfeccionar nuestro *socket* tanto como queramos.

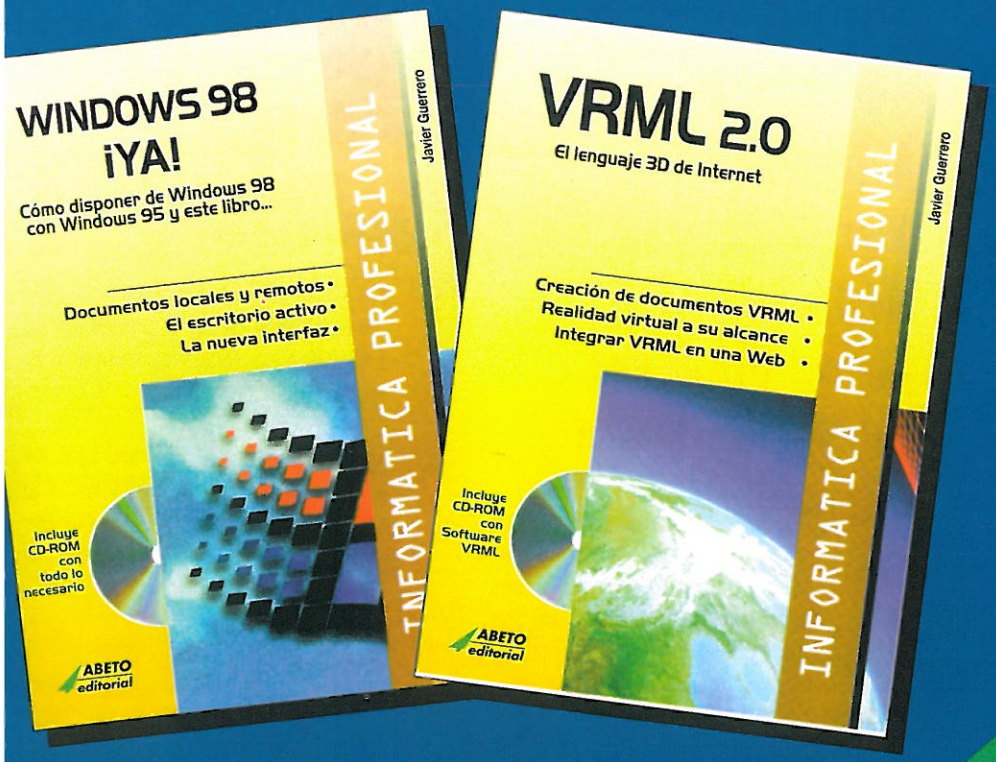
Hemos visto, además, cómo crear un cliente *echo*, que nos permita verificar que funciona la conexión TCP/IP con un servidor y por ende que funciona nuestro *socket*. En próximos números estudiaremos el protocolo HTTP y, también, cómo crear nuestra propia aplicación cliente HTTP, es decir, nuestro propio (y pequeño) navegador de WWW.

PARA MOVERTE EN

INFORMÁTICA COMO

PEZ EN 

EL AGUA



1995 ptas. IVA incluido

Incluyen CD-ROM de regalo

Ya a la venta en quioscos y librerías.

ABETO editorial

c/ Aragoneses, 7 • 28108 Alcobendas (Madrid)
Tel.: (91) 661 42 11* • Fax: (91) 661 43 86

JAVA

Servlets (I): La cara oculta de la web

Juan Carlos Sastre Iglesias (jcsi@geocities.com)

Java al igual que la luna tiene una cara oculta que pocos conocen. Su cara visible son los applets, que amplían de forma sorprendente la interactividad de las páginas Web, y su cara oculta son, sin lugar a dudas, los servlets, menos espectaculares, pero con una importancia fundamental.

La parte más conocida de Java son los *applets*. Estos no son más que programas que se pueden cargar a través de una red y que se ejecutan de igual manera en cualquier plataforma, todo ello gracias a las potentes características de Java. Hasta hace poco, Java se utilizaba básicamente para dotar a las páginas Web de una mayor interactividad mediante los *applets*, y por tanto sólo actuaba sobre el lado cliente. Pero el lado servidor también puede beneficiarse de todas las ventajas que ofrece Java, gracias a los *servlets*.

Java se ha venido usando básicamente para dotar a las páginas Web de una mayor interactividad

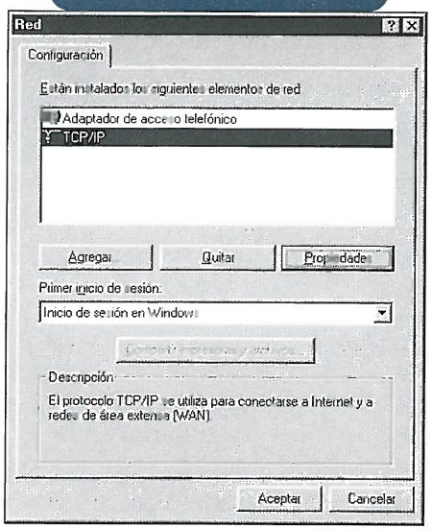
Los *servlets* se diferencian de los *applets* fundamentalmente en que se ejecutan en el servidor y no presentan ningún tipo de interfaz gráfica puesto que se encargan de hacer el trabajo oculto. En este curso abordaremos la programación de *servlets* en un entorno cliente-servidor basado en http. De hecho este es uno de sus aspectos más interesantes, y es por lo que muchos programadores que hasta ahora utilizaban CGI's, están utilizando *servlets*. De hecho, hasta ahora los CGI's eran el único medio de proporcionar interacción entre el cliente y el servidor. Un ejemplo muy común de uso de los CGI's son los típicos formularios que el usuario ha de rellenar con sus datos personales, que posteriormente pasan a formar parte de una base de datos.

¿Sustituirán a los CGI's?

A continuación se enumeran algunas de las razones que evidencian este hecho:

- Se programan utilizando un API estandarizada Java, y por tanto se benefician de todas las ventajas de este potente lenguaje.
- Consumen menos recursos que los CGI's y los Fast-CGI's. Por ejemplo no es necesario cargarlos en memoria tantas veces como peticiones de ejecución existan.
- Se pueden cargar indiferentemente y de forma transparente, tanto desde un disco local como desde una dirección remota. Responden a la nueva filosofía de software distribuido.
- Los *servlets* pueden comunicarse entre sí, y por tanto, es posible una

Figura 1.



reasignación dinámica de la carga de proceso entre diversas máquinas. Es decir, un *servlet* podría pasarle trabajo a otro *servlet* residente en otra máquina.

- Poseen una estructuración en las fuentes que resulta mucho más clara que los CGI's (especialmente si esta comparación se efectúa con respecto a PERL).

Los servlets se ejecutan en el servidor y no presentan ningún tipo de interfaz

- Se pueden reutilizar CGI's ya hechos, incrustándolos en *servlets*.

- Es posible utilizarlos en servidores tan populares como el Apache, el Fastracker o bien el Internet Information Server.

Los *servlets* son la mejor opción a partir de ahora para desarrollar aplicaciones para servidores WEB. Trabajan de forma transparente con las filosofías GET y POST de los formularios HTML, y también como es lógico, se entienden perfectamente con los *applets*, pero también se pueden entender con clientes programados en cualquier otro lenguaje.

Aplicaciones

Junto a los *servlets* es posible implementar por ejemplo:

- Típicos sistemas Middleware que hasta ahora únicamente se implementaban con CGI's. Para consultar las Bases de Datos, los *servlets* pueden utilizar JDBC (Java Data Base Connection), lo que les permite extraer información de cualquier sistema de Base de Datos.

Los servlets son la mejor opción para desarrollar aplicaciones para servidores Web

- Automatización de un sistema de recepción y publicación de información. Por ejemplo podríamos montarnos una simple estación meteorológica que permitiese acceso a su información mediante una página Web. Por un lado tendríamos un servlet que recolectaría la información de los diversos tipos de sensores y la almacenaría en bases de datos, y por otro lado, un servlet que se encargaría de presentar esta información en función de las peticiones del cliente basándose en estas mismas bases.

- Control de la recepción de e-mails, y sistemas de news, chats... Sería recomendable que se recordara a este respecto que Java es un lenguajes especialmente indicado para la programación utilizando los protocolos TCP/IP (SMTP, NNTP, conexiones socket, etc).

- Obviamente pueden generar páginas HTML en el cliente de una manera dinámica.

- Dado que pueden manejar múltiples peticiones concurrentemente, es posible implementar aplicaciones colaborativas, como por ejemplo una aplicación de videoconferencia.

- Se dispone de la posibilidad de implementar agentes inteligentes capaces de interactuar entre ellos.

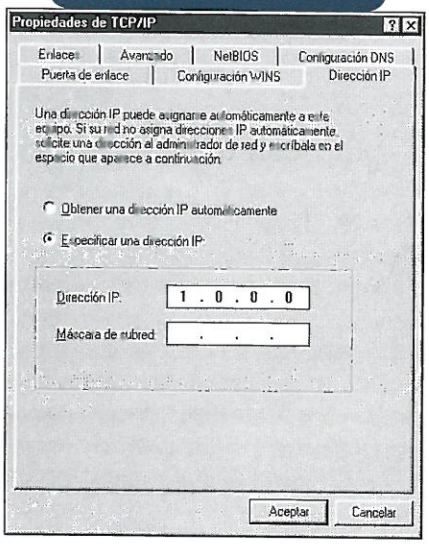
- Y otras posibilidades de implementación de aplicaciones que habría que añadir a las enumeradas.

Manos a la Obra

Para seguir adecuadamente la serie es necesario configurar un entorno de trabajo que resulte adecuado para poder experimentar con *servlets*. Existen varios puntos fundamentales:

- Sistema operativo.
- Servidor Web.
- Versión del Java Development Kit.
- Configuración del TCP/IP.
- Configuración de las variables de entorno.

Figura 2.



- **Sistema Operativo**

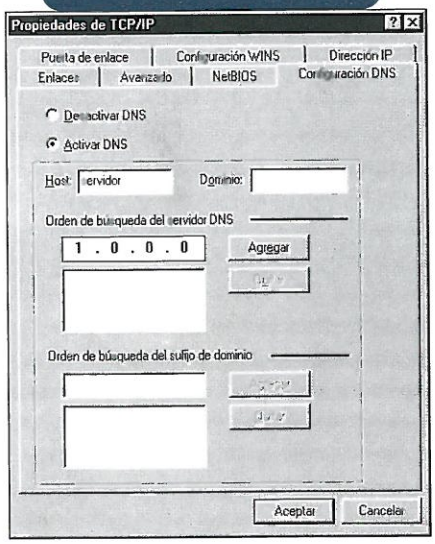
  A lo largo de todo el curso se asumirá que se trabaja sobre una máquina con Windows 95. En la elección del sistema operativo se ha tenido en cuenta que la mayoría de las personas que siguen esta serie dispondrán y conocerán este popular sistema operativo.

Se utilizará el sistema operativo Windows 95 por ser uno de los más utilizados y populares

- **Servidor Web**

  Para un seguimiento ideal de este curso se ha de disponer del Java Web Server (a partir de ahora se abreviará como JWS), que presenta como característica más destacada el soporte nativo de *servlets*. No se va a hacer un estudio de sus características y posibilidades, simplemente se indicará lo necesario para hacer funcionar los *servlets*. Es posible bajarse la última versión en la siguiente dirección:

Figura 3.



http://www.javasoft.com/products/java-server/webserver/index.html

- **Versión del Java Development Kit (JDK)**

  La elección de la versión del JDK es fundamental. Existen dos opciones que son el JDK 1.0.x ó el JDK 1.1.x. Como se verá en los próximos capítulos, se harán cosas con JDBC habiendo optado por el JDK 1.1.x. Es posible bajarse la última *release* en la siguiente dirección:

  http://www.javasoft.com/products/jdk/1.1/

- **Configuración del TCP/IP**

  Para poder trabajar de forma cómoda será necesario configurar los parámetros TCP/IP de Windows 95, de forma que parezca que nuestra máquina forma parte de Internet aunque no estemos conectados. Para ello se seguirán los pasos enumerados a continuación:

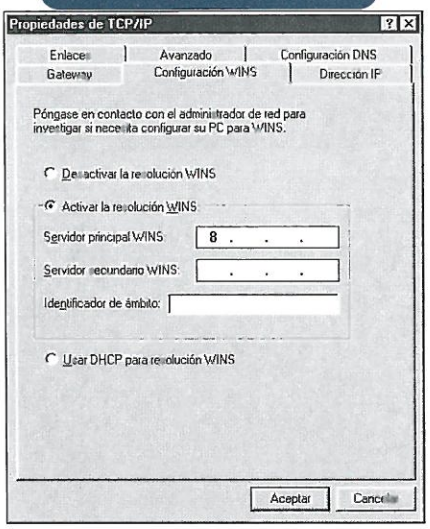
La configuración se realiza a partir de los parámetros TCP/IP, de manera que parezca que estamos conectados

1º Haciendo doble clic sobre el icono de Red que se encuentra en el Panel de Control, se pulsará sobre propiedades el elemento TCP/IP (figura 1).

2º En la pestaña Dirección IP se indicará una IP como por ejemplo 1.0.0.0 (figura 2).

3º En la pestaña Configuración DNS se activará el servicio DNS y se especificará un nombre de Host. Asimismo se especificará

Figura 4.



la IP donde debe buscarlo (se escribirá y se pulsará sobre el botón Agregar) (figura 3).

Conviene recordar que se tendrá que desactivar esta configuración en el caso que se desee conectar a Internet.

Configuración de las variables de entorno

Se tendrá que añadir a la variable de entorno CLASSPATH la localización de las clases necesarias para poder compilar los *servlets*. Esto es así porque el API de los *servlets* no viene con el API estándar del JDK (es seguro que con el JDK 1.2 ya vendrá incluida). Suponiendo que el JWS se instala en el directorio por defecto, se tendrá que añadir una línea de este estilo:

```
set CLASSPATH=.;c:\JavaWebServer1.0.3\lib\classes.jar;%CLASSPATH%
```

Una vez modificado el AUTOEXEC.BAT es necesario reiniciar la máquina (no se cargarán las últimas modificaciones si se selecciona Reiniciar el Equipo con la tecla Shift pulsada).

La librería de Acceso (IV)

Jorge F. Delgado Mendoza (ernar@convex.es)

A lo largo de tres artículos, hemos ido descubriendo, poco a poco, las enormes posibilidades que hay detrás del concepto de Sistema Distribuido.

En el primero de los artículos de la serie, analizamos las carencias de las aproximaciones clásicas al problema de las aplicaciones cuyo ámbito de operación se extiende más allá de la frontera de una máquina única.

Los problemas que se le plantean al programador a la hora de utilizar varias máquinas para ejecutar su aplicación son muy numerosos. Por un lado, no todas las máquinas del sistema son iguales, ni tienen por qué tener los mismos procesadores o sistemas operativos. Por otro lado, la configuración de la red de explotación puede variar dinámicamente con el tiempo, por lo que es necesario planificar la adición y sustracción de las máquinas que la componen.

Este tipo de situaciones conducen a la creación de soluciones poco eficientes desde varios puntos de vista:

Desde la perspectiva de la *escalabilidad*, encontramos aplicaciones que, comportándose espléndidamente sobre un pequeño número de ordenadores, se vuelven prácticamente inútiles cuando se añade una decena de nuevas máquinas. Sistemas incapaces de crecer de acuerdo con serie de necesidades de los usuarios, por culpa de las limitaciones existentes en su implementación. Aplicaciones diseñadas para un gran número de unidades de proceso desperdiciando capacidad de cálculo cuando apenas existen servidores en la red.

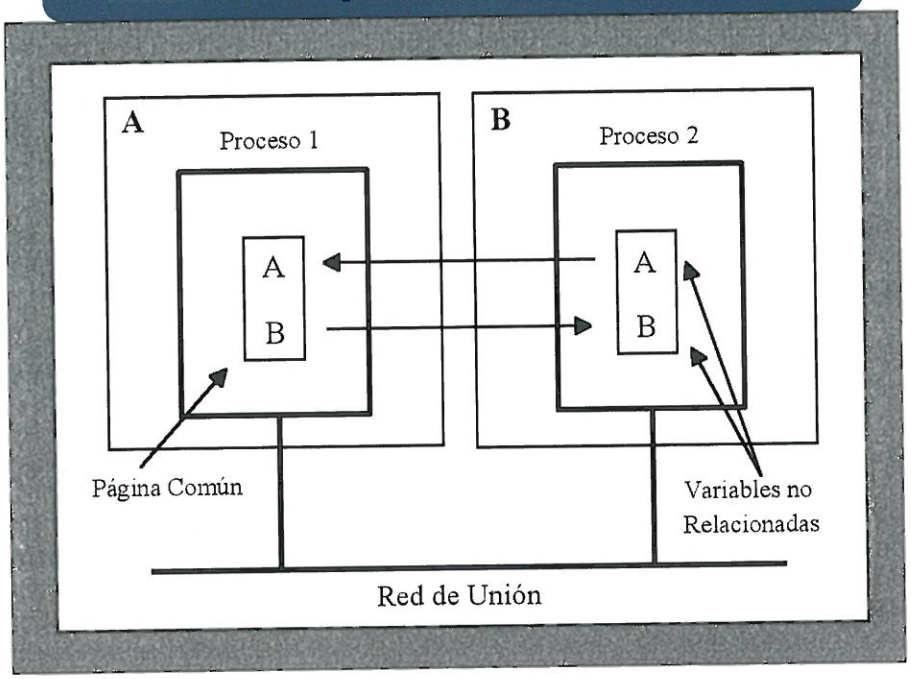
Si analizamos la *portabilidad*, nos encontramos con sistemas tan ligados a una arquitectura, que son prácticamente imposibles de utilizar en otro tipo de máquinas. Aplicaciones capaces de correr en diferentes Sistemas Operativos que son, a su vez, incapaces de entenderse entre sí. Programadores que, acostumbrados a un tipo de entorno de desarrollo, son incapaces de portar las soluciones a otros sistemas, etc...

Finalmente podemos citar la *complejidad*. Si cada vez que queremos crear una aplicación capaz de ejecutar en varias máquinas simultáneamente, tenemos que construir un nuevo protocolo de comunicaciones, los algoritmos de trasiego de información, etc.., el ciclo de desarrollo se dispara. Además se corre el peligro de perder el objetivo fundamental de la aplicación, enmascarado detrás de una maraña de mensajes que circulan de ordenador a ordenador.

Como veíamos en el primer artículo, el concepto de *Sistema Distribuido* permitía paliar todas estas carencias de una manera elegante, simple y eficaz. Toda la complejidad del sistema de comunicaciones se concentraba en el entorno distribuido que proporciona al desarrollador de

A pesar de tener ya completamente diseñado el corazón de nuestro Sistema Distribuído, todavía nos falta una parte esencial del mismo; debemos dotarlo de un método de acceso que permita construir aplicaciones sobre él.

Figura 1.- La falsa compartición es ocasionada por el trasiego constante de un bloque de memoria entre dos o más máquinas del sistema. El bloque contiene dos variables accedidas con mucha frecuencia en ambas máquinas, lo que ocasiona que los datos tengan que actualizarse muy a menudo a través de la red. Este comportamiento hace disminuir el rendimiento global del sistema en varios órdenes de magnitud.



una API de máquina virtual única. Esta abstracción le permite concentrar toda su atención en resolver el problema, como si se tratase de un sistema formado por un solo ordenador.

A lo largo de este primer artículo, analizamos los diferentes aspectos que se debían tener en cuenta a la hora de desarrollar un Sistema Distribuido. Enumeramos las diferentes alternativas que podían utilizarse para su implementación, justificando la oportunidad de unas u otras en función del tipo de problema a resolver.

Una vez analizadas las diferentes opciones, decimos construir un sistema ejemplo. Todo el contenido del segundo artículo se dedicó a sopesar las diferentes opciones, con el fin de llegar a una arquitectura de Sistema Distribuido capaz de cubrir todas nuestras necesidades. Los requisitos del sistema eran:

Sencillez en su implementación. Al ser un sistema concebido principalmente con un propósito didáctico, hemos preferido sacrificar rendimiento y características no esenciales a cambio de poder presentar los conceptos fundamentales de un entorno de esta índole.

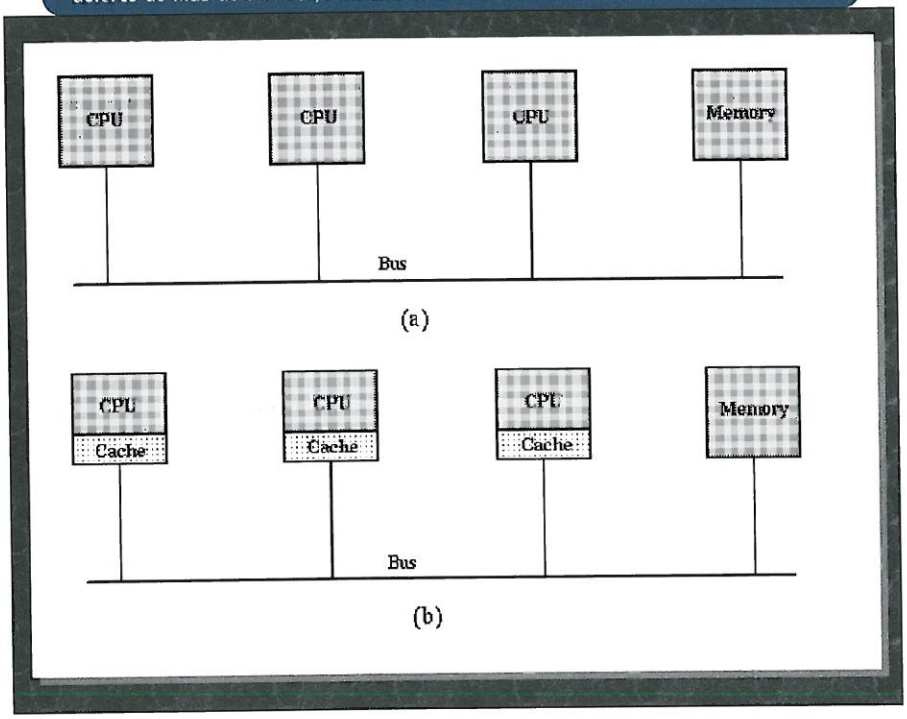
Universalidad. A pesar de ser un sistema sencillo, nuestra intención era no dejar de lado ninguno de los conceptos fundamentales sobre los que se ha de basar la construcción de un sistema profesional.

Rendimiento. Para que un sistema sea útil, es necesario poder construir aplicaciones sobre él. Esto nos obliga a que los tiempos de respuesta del sistema sean suficientemente rápidos como para soportar accesos interactivos.

Sin embargo, el motivo más importante era, sin duda, dotar al programador de un sistema con una interfaz de fácil manejo, que le ocultara la complejidad del protocolo de comunicaciones. Presentar una API de *Máquina Virtual Única*.

Teniendo en cuenta estos principios de diseño, dedicamos el resto del segundo

Figura 2.- Si imaginamos un sistema donde varios ordenadores acceden a una memoria compartida, cada dato que se necesite se adquirirá mediante la red. Para paliarlo, se utiliza una memoria local que guarde los valores accedidos recientemente. Unido a un sistema de control que permita mantener la coherencia entre las diferentes memorias locales, la propiedad de localidad de referencia hace que el sistema obtenga tasas de acierto de más de un 90%, reduciendo así los accesos a la red a uno de cada diez.



IBM
Soluciones para nuestro pequeño mundo

artículo a analizar los diferentes puntos clave de la arquitectura de un Sistema Distribuido.

Decidimos utilizar como método para comunicar los procesos residentes en las diferentes máquinas, el paradigma conocido como *RPC* o Llamada a Procedimiento Remoto. (del inglés *Remote Procedure Call*). La sencillez de su utilización compensa más que de sobra la pequeña pérdida de rendimiento en la que incurrimos con su utilización. Además, al existir para todo tipo de arquitecturas y Sistemas Operativos, va a simplificar enormemente la tarea de construir un entorno heterogéneo.

A partir de los fundamentos teóricos expuestos en el primer artículo, analizamos los diferentes *Modelos de Consistencia*. Éstos no son más que un conjunto de reglas que nos van a permitir tratar un espacio de memoria. El espacio está distribuido entre todos los ordenadores del sistema como si fuera un único conjunto de datos centralizado.

El modelo elegido fue una variante del sistema de *Consistencia Secuencial*. A favor tenía su sencillez de compresión, -y por tanto de utilización- por parte del programador que utilizaba el Sistema Distribuido. En su contra tenía por otro lado la relativa falta de eficiencia del modelo frente a metodologías de coherencia más sofisticadas.

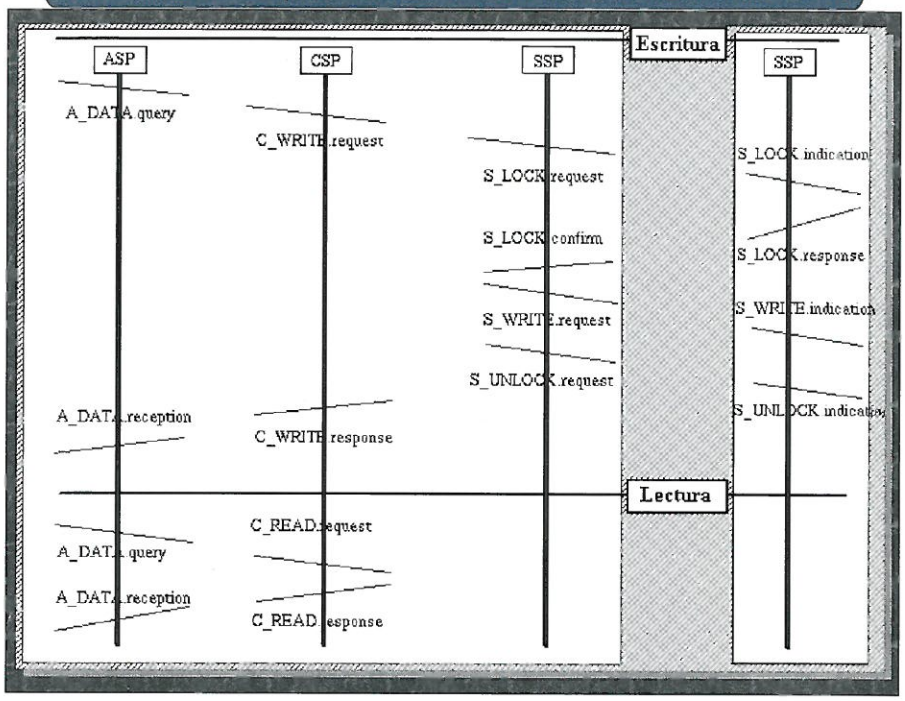
En cuanto al modelo de memoria, por razones de eficiencia, se decidió empezar a utilizar un sistema de Variables Compartidas. En este modelo, sólo una pequeña cantidad del espacio de memoria es común a todas las máquinas, mientras que la mayor parte de los datos -que no necesitan ser accedidos de manera distribuida- se manipulan localmente. A cambio de una pequeña pérdida de generalidad, (no es posible utilizar todo el campo de variables de la misma manera), se matan dos pájaros de un tiro, ya que se aumenta el rendimiento general y se minimiza el riesgo de *falsa compartición*.

Una vez determinado el modelo de memoria a emplear, estudiamos diferentes estrategias de replicación. A pesar de que añade una complicación extra al sistema, su utilización es enormemente generalizada. El incremento de rendimiento que se obtiene con un sistema adecuado de caches, tiene como resultado que sea prácticamente imprescindible incorporarlo a cualquier entorno de estas características.

Finalizado el análisis que se ha venido exponiendo, se está en disposición de determinar las primitivas de comunicaciones que deberían circular por la red, que interconecta los diferentes equipos. Una vez se han definido, su secuencia de utilización determinará el protocolo del sistema.

El protocolo elegido, tanto para lecturas como para escrituras, se puede observar fácilmente en la figura número tres. Los motivos que nos han llevado a su elección se detallan en el segundo y tercer artículo de la serie.

Figura 3.- Protocolo de comunicaciones entre dos máquinas. La figura muestra la secuencia de mensajes necesaria para realizar peticiones de lectura y de escritura al Sistema Distribuido. La aplicación usuaria accede al mismo a través de la interfaz ASP.



En el tercer artículo nos concentramos en la construcción del componente fundamental del Sistema Distribuido, el *RTS*[1]. Construido como un demonio, el cual se ejecuta constantemente en el ordenador, su misión consistía en mantener la coherencia de los datos residentes en los distintos PCs.

Sin embargo, todo este trabajo sería inútil si nadie puede utilizarlo para construir aplicaciones. Uno de los requisitos *fundamentales* del sistema era dotar al usuario, en este caso al programador que construye aplicaciones sobre varias máquinas, de una API de fácil utilización.

Para conseguir este objetivo, hemos decidido integrar todas las primitivas de acceso al sistema en una librería, llamada:

libdsm.a

El presente artículo, cuarta entrega de la serie, se va a dedicar por entero a la tarea de construir un interfaz de acceso homogéneo y de fácil utilización. Para ello, vamos a seguir la estructura seguida en anteriores entregas.

Comenzaremos determinando los requisitos necesarios para poder llegar a realizar esta tarea, enumerando las herramientas y aplicaciones que serían necesario instalar en nuestro ordenador, así como los conocimientos que se suponen al lector del artículo.

Finalmente, dejaremos una puerta abierta a posibles mejoras en el sistema y volveremos a mencionar la bibliografía que se puede consultar para profundizar en el tema de la programación distribuida.

Requisitos

Conocimientos: para facilitar la portabilidad entre diferentes arquitecturas, todo el sistema se ha codificado en lenguaje de programación 'C'. La librería no va a ser una excepción, por lo que un conocimiento relativamente profundo del mismo es imprescindible.

A pesar de que apenas se toca el tema, sería útil un conocimiento relativo del funcionamiento de las Llamadas a Procedimiento Remoto. En el tercer artículo de la serie, aparecido en el número anterior, explicamos paso a paso la utilización de las mismas.

Para entender lo que se desea conseguir, es necesario tener una idea clara de los conceptos en los cuales se basa la construcción de un sistema de estas características. Tanto la bibliografía mencionada al final de este artículo, como los dos primeros artículos de la serie presentan de una manera didáctica los conceptos a los que nos referimos.

Software: necesitaremos las siguientes herramientas instaladas en todos los ordenadores del sistema:

- El compilador GCC, versión 2.7.0 ó superior.
- La herramienta MAKE.
- El *software* de RPC's.
- El *software* de TCP/IP.
- El *RTS* del Sistema Distribuido[2].

Además, es necesario que el demonio de RPC, llamado *portmapper*, se encuentre activo siempre que queramos utilizar el sistema. Para activarlo, podemos hacerlo a mano como *root* cada vez que lo necesitemos, aunque lo ideal es añadir al fichero de configuración de red las siguientes líneas:

```
# Start the SUN RPC Portmapper.
if [ -f ${NET}/rpc.portmap ]; then
  echo -n " portmap"
  ${NET}/rpc.portmap
fi
```

El fichero de configuración al que nos referimos es el que se ejecuta al iniciar el sistema y el *path* completo hasta él es:

/etc/rc.d/rc.inet2

Una vez realizada esta operación, cada vez que vayamos a arrancar el ordenador, dispondremos de los servicios asociados a las RPC's.

A pesar de que toda la implementación descrita se ha realizado sobre Linux

IBM
Soluciones para nuestro pequeño mundo

2.0.30, el entorno completo ha sido probado con éxito, aunque con pequeñas modificaciones, en los siguientes Sistemas Operativos:

- *Solaris 2.4 (Sun UltraSparc).*
- *Irix 5.x (Silicon Graphics Indy-R4400).*
- *HP-UX.*

Documentación: Como ya se ha mencionado anteriormente, una copia de los dos primeros artículos sería especialmente útil si se pretende llegar a comprender el sistema que vamos a acabar de construir. Sin embargo, el tercer artículo es imprescindible, ya que contiene el código necesario para poder construir el RTS, y por tanto, el sistema.

Objetivos

El objetivo de este artículo es de doble. Por una lado, vamos a construir la interfaz del Sistema Distribuido con el usuario, mientras que por el otro aprenderemos cómo se puede llegar a crear e instalar una librería en *Linux*.

Si recordamos el diagrama de bloques que se había definido en el segundo artículo de esta serie, podemos observar cómo la interfaz ASP permitía comunicar las aplicaciones del usuario con la *entidad de acceso* del sistema.

Por lo tanto, nuestra tarea consistirá en implementar la parte del interfaz que va a ver el usuario, adecuándola a la que ya realizamos en el tercer artículo de la serie.

Una vez realizado todo el código necesario, habremos de compilarlo compactando el código objeto en una librería que podamos reutilizar cada vez que queramos. Para completar esta tarea, aprenderemos como crear una librería en *Linux*.

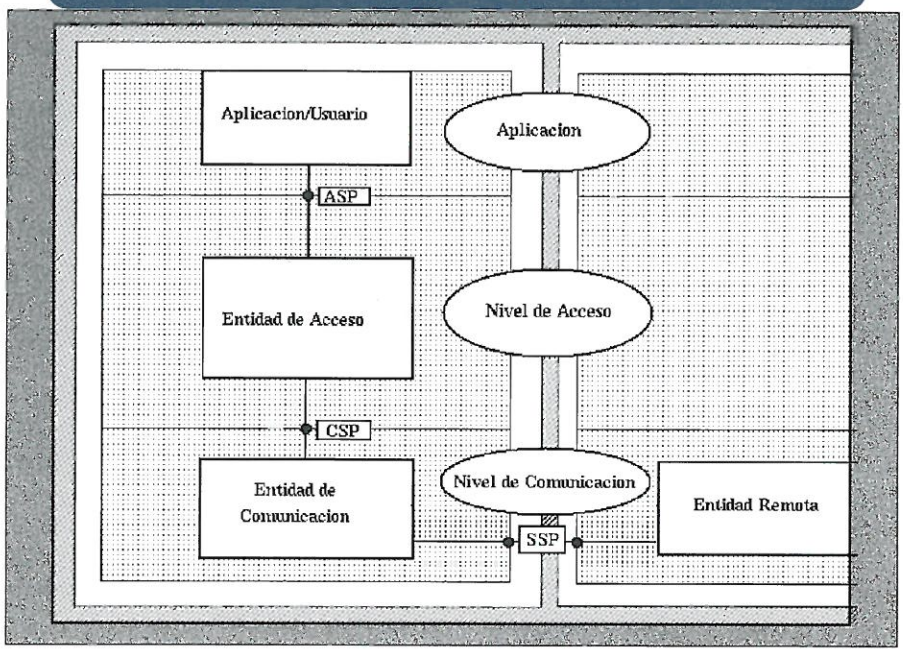
Recordando la Interfaz

Con el objetivo de poder acceder al espacio de memoria de nuestro Sistema Distribuido, vamos a necesitar una o más funciones de conexión con el servidor.

En el tercer artículo, implementamos la solución utilizando una única función de contacto. Esta función contenía un paquete de datos, organizado como una estructura, en el cual introducíamos toda la información relativa al mensaje.

El paquete de datos, el cual respondía al tipo *Frame*, tenía cuatro campos fundamentales. El significado que tienen los mismos se puede ver en la Tabla 1.

Figura 4.- División en capas del Sistema Distribuido. El objetivo de la librería que vamos a crear a lo largo de este artículo, proporcionará al usuario las primitivas de la interfaz ASP.



Cada vez que queramos mandar un mensaje, deberemos rellenar los tres campos relevantes de la estructura *Frame* y utilizar la función que hayamos definido para conectarnos con el servidor o RTS.

Implementación de la Librería

Con la finalidad de crear la librería vamos a tener que utilizar algunos de los ficheros que generamos al crear el servidor de la memoria compartida.

Estos ficheros son los que se exponen a continuación:

- **proto5.h** Cabeceras asociadas a los cabos de la RPC que habitualmente son utilizadas para comunicar los clientes con el servidor.
- **proto5_xdr.c** Código encargado de mantener la integridad del formato que circula por la red.
- **proto5_clnt.c** Se trata del cabo del cliente de la RPC.

Además de estos archivos, hemos de crear otros tantos que se ocuparán de proporcionar la funcionalidad requerida:

- **dsm.h:** se encargará de proporcionar el interfaz a las aplicaciones de las que dispone el usuario, mediante definiciones de todos los parámetros accesibles de la librería.
- **client_if.c:** define el interfaz de comunicación con el sistema. Contiene la función *DSM_Contact()*.
- **dsm.c:** contiene la función de inicialización del sistema *DSM_Init()*.

A continuación describimos los detalles de implementación de las dos funciones relevantes, siempre desde el punto de vista del usuario.

- **Función DSM_Init**

La función *DSM_Init()* se encarga de preparar el sistema para su utilización.

Con este fin, se va a crear un enlace de comunicaciones con el RTS local del sistema de variables distribuidas, obteniendo así un identificador único del canal. Este identificador se devuelve a nuestro programa.

A partir de su invocación, todas las tramas de datos enviadas al servidor del Sistema Distribuido, utilizarán este manejador como argumento.

La implementación es enormemente sencilla, ya que nos limitamos a incluir la cabecera asociada a la RPC, que se generó con *rpcgen* al crear el RTS; y a realizar una llamada a la librería de comunicaciones para que cree un cliente de acceso.

Este cliente de acceso utilizará el protocolo de comunicaciones *TCP* que sirve para comunicarse con la máquina local, de ahí la utilización de *localhost* como primer argumento.

- **Función DSM_Contact**

El objetivo de esta función es enviar un mensaje al Sistema Distribuido. Antes de invocarla, habremos rellenado la estructura que pasamos como argumento, definiendo el tipo de operación que deseamos realizar y añadiendo los valores necesarios para completarla con éxito.

Tabla 1.- Significado de los campos de la estructura que se utiliza para enviar mensajes al servidor.

Campo	Significado
Frame.function:	Determina el tipo de operación a realizar.
Frame.coordinate:	Indica la posición de la memoria compartida sobre la que realizar la operación.
Frame.value:	Valor de la posición de memoria accedida.
Frame.auxiliar:	Valor auxiliar, utilizado para mantenimiento interno.

Su implementación es, como en el caso anterior, muy sencilla.

Comenzamos incluyendo los ficheros necesarios para poder utilizar la librería de RPC, así como la cabecera obtenida mediante *rpcgen*.

Una vez realizada esta tarea, declaramos el manejador de la conexión RPC, de tal manera que podamos utilizar el dato obtenido mediante *DSM_Init()*.

Finalmente, llamamos a la función *contact_1*. Esta rutina, generada automáticamente por *rpcgen*, nos permite contactar con el cabo asociado al servidor del Sistema Distribuido.

- **Cabecera de la Librería: dsm.h**

En la cabecera de la librería hemos de incluir todos aquellos datos que van a ser de utilidad al usuario de nuestro Sistema Distribuido.

Estos datos a los que nos referimos, están incluidos y diferenciados en tres bloques muy distintos: declaración de constantes, definición de variables y estructuras y declaración de funciones:

Las constantes que debemos declarar incluyen los niveles de error de la librería, los valores asignados a las distintas operaciones, -lectura, escritura, etc..- así como la inclusión del código asociado a las RPC, necesario para poder utilizar los protocolos necesarios.

La única estructura que tenemos que declarar es la que va a contener el mensaje: *Frame*. A continuación, y para hacer más legible el código, definimos un tipo con el mismo nombre, de tal manera que podamos definir estructuras de este tipo mediante la línea.

```
Frame <nombre de la variable> ;
```

Para concluir con la construcción de la cabecera de nuestra librería, añadiremos las declaraciones de las dos funciones que se pueden utilizar en este caso: *DSM_Init()* y *DSM_Contact()*.

Figura 5.- La función DSM_Init() se limita a obtener un manejador de comunicaciones asociado al servidor o RTS del Sistema de Memoria Compartida. Es imprescindible que sea llamada antes de comenzar a realizar accesos a la misma. Aunque es posible obtener varios manejadores, el resultado de enviar datos a través de uno u otro es exactamente el mismo.

```
#include "proto5.h"

CLIENT *
DSM_Init (void)
{
    return clnt_create("localhost",
                       RSERVERPROG,
                       RSERVERVERS,
                       "tcp");

}
```

Instalación de la Librería

Lo primero que debemos hacer para poder utilizar *libdsm.a* en nuestros programas es tenerla instalada en el sistema. Al ser una librería creada por el usuario, normalmente la colocaremos en:

```
/usr/local
```

El código incluido en el CD-ROM de la revista se puede copiar en cualquier directorio del sistema. Dado que para instalarla vamos a tener que ser *root*, aprovecharemos para poner el código fuente en */usr/src/lib_dsm*.

Una vez hemos logrado copiar el código copiado en el sistema, nos introducimos en el directorio y tecelamos el siguiente comando:

```
make
```

Al cabo de unos minutos, la compilación habrá terminado. Además de los clásicos ficheros *.o, habrá aparecido la librería en sí bajo el nombre:

```
libdsm.a
```

Creada la librería, basta con que la copiemos en el directorio adecuado mediante el comando:

```
cp libdsm.a /usr/local/lib/.
```

A continuación copiamos el fichero de la cabecera en su lugar correspondiente utilizando el comando:

```
cp dsm.h /usr/local/include
```

Una vez llevados a cabo estos pasos, para concluir la instalación, forzaremos a que el sistema vuelva a leer el conjunto de librerías disponibles. Para ello utilizaremos el comando:

```
ldconfig
```

Acabado este proceso el sistema nos permitirá crear programas contra nuestra recién creada librería.

Más Allá

La relativa sencillez de la librería no nos deja prácticamente campo para introducir mejoras en el sistema. Sin embargo, existen dos posibles expansiones en cuanto a la funcionalidad de la misma:

Se dispone de la posibilidad de crear una *librería de enlace dinámico*. De esta manera la utilización de memoria por parte de las distintas aplicaciones concurrentes será más racional, evitando cargar toda la librería en cada ejecutable.

Simplificar la utilización de la función *DSM_Contact*, incluyendo una serie de macros en la cabecera de la librería. Éstas se encargarían de rellenar la estructura del mensaje en función del tipo de operación a realizar. De esta manera podríamos tener una macro para realizar lecturas, otras para escrituras, etc...

Figura 6.- La función DSM_Contact() se encarga de contactar con el servidor del Sistema Distribuido. Para ello utiliza el cabo generado a partir de la especificación de protocolo de comunicaciones generado en el artículo anterior mediante rpcgen.

```
#include <rpc/rpc.h>
#include <stdio.h>
#include "proto5.h"

/* Manejador de la RPC */
extern CLIENT *handle;

int
DSM_Contact(argument)
     Frame argument;
{
  Frame * args;
  args = &argument;
  return *contact_1(args,handle);
}
```

Referencias bibliográficas

La colección de libros y artículos que se va a enumerar a continuación, puede ser interesante para todo aquél que tenga la intención de profundizar en el campo de los Sistemas Distribuidos:

- Andrew S. Tanenbaum, *"Distributed Systems"*, Ed. Prentice-Hall International.
- Douglas E. Comer y David L. Stevens, *"Internetworking with TCP/IP Vol III"*, Ed Prentice-Hall International.
- Andrew S. Tanenbaum, *"Parallel Programming Using Shared Objects and Broadcasting"*.
- Nicholas Carriero y David Gelernter, *"How to Write Parallel Programs: A Guide to the Perplexed"*. ACM Computing Surveys, Vol 21, Nº3
- Henri E. Bal, *"A Comparative Study of Five Parallel Programming Languages"*, Proceedings of the EurOpen Spring 1991 Conference on Distributed Systems.
- C. L. Hartley y V. S. Sunderam, *"Concurrent Programming with Shared Objects in Networked Environments"*.

Conclusiones

En este artículo, el cuarto de la serie, hemos alcanzado nuestro objetivo de crear un Sistema Distribuido. Para ello hemos aprendido los conceptos básicos, y los que no se consideran tan básicos. Hemos realizado un análisis del sistema a construir, llegando a una arquitectura capaz de satisfacer nuestras necesidades. Y finalmente hemos implementado, tanto el sistema en sí como una interfaz cómoda y sencilla que facilite la tarea del desarrollador.

En el próximo artículo cambiaremos los papeles. En vez de ser los diseñadores de un entorno distribuido, preocupados por el rendimiento de los protocolos, la coherencia de la memoria, etc.., vamos a convertirnos en el programador que utiliza el sistema de memoria compartida para crear una aplicación.

Para ello vamos a crear un pequeño universo virtual muy simple, accesible mediante varios ordenadores simultáneamente. Sin ningún ánimo de emular al famoso *Quake*, cuya secuela acaba de salir al mercado, vamos a intentar mostrar las ventajas que un Sistema Distribuido puede aportar a un programa conceptualmente similar.

El autor

Jorge Delgado Mendoza es miembro de XFree86 desde el año 1994. Ingeniero Superior de Telecomunicaciones por la UPM, trabaja actualmente en Convex SuperComputers. Puede ser contactado por correo electrónico en **ernar@convex.es**.

[1] *RTS: Run Time System*

[2] El RTS del Sistema Distribuido se describió en el número anterior de la revista. El lector podrá encontrar las fuentes que pertenecen al código en el CD-ROM de la misma.

LINUX

Configuración de clientes X con Motif Windows Manager

Arturo Álvaro

A lo largo de este artículo podremos ver una serie de aspectos concretos referidos a las técnicas orientadas a la configuración de los clientes X y sus aplicaciones.

Nuestra plataforma

En cuanto al sistema operativo podemos decir que la distribución de Unix que elegiremos para nuestros desarrollos se basará en Linux, ya que posee una potencia similar a la de otros Unix del mercado, al tiempo que presenta la ventaja de ser un sistema de distribución gratuita, ampliamente utilizado y soportado.

La distribución de Unix elegida para nuestros desarrollos se basará en Linux, presentando la ventaja de ser un sistema de distribución grautuita

Su obtención es bastante sencilla: lo hemos encontrado en números de esta misma publicación, y tenemos siempre disponibles las últimas versiones a través de la red Internet.

El entorno gráfico: Obviamente estamos hablando de X-Windows. Al usar Linux como sistema operativo, tendremos que ceñirnos a la implementación existente sobre el mismo.

Sin embargo, gracias a la idea de estandarización existente en la filosofía Unix, la base de la configuración es la misma para todos los sistemas, con lo que si tenemos que trabajar con cualquier otro, o migrar nuestras aplicaciones en el futuro, sólo encontraremos problemas con algunos pequeños detalles de fácil solución.

El sistema Linux, además de poseer los tradicionales clientes X-Windows, como pueden ser Xclock, Xterm o Bitmap, trae consigo gestores de ventanas como uwm, fwm o mwm, que son intermediarios entre los clientes y el servidor. Estos manejadores de ventanas controlan, sobre todo, la apariencia de los programas cliente al ser ejecutados en el entorno gráfico.

Ya sabemos que entre los clientes y el servidor, existen lo que se denominan "peticiones", (crear ventana, iconizar, etc) ,y "respuestas", que son generadas por el servidor de forma asíncrona.

En nuestro caso vamos a aprender sobre las técnicas de configuración bajo el

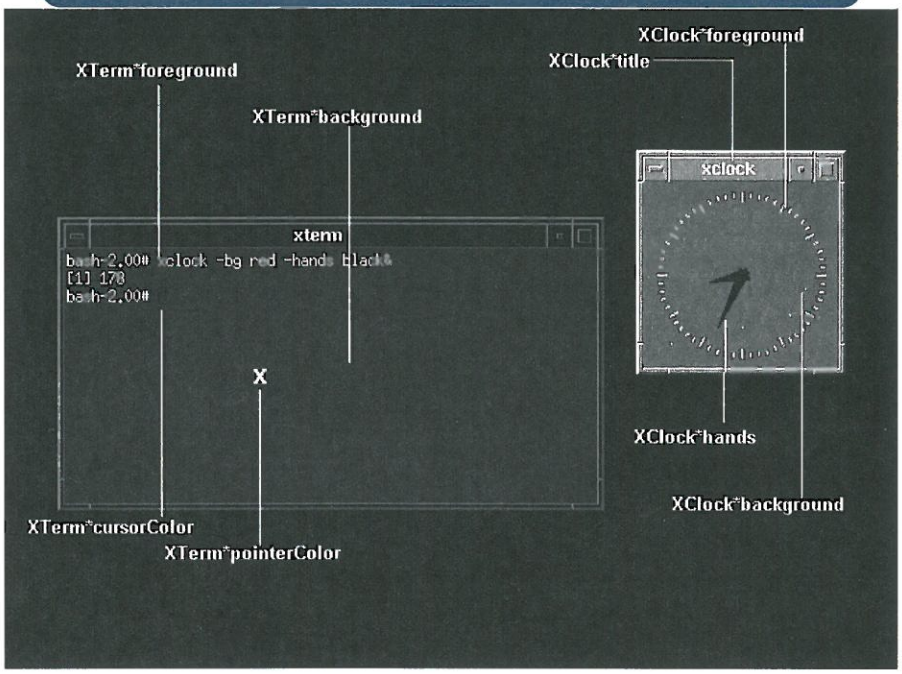
En los últimos años, debido al desarrollo de librerías de programación para las plataformas X-Windows en lenguajes de alto nivel, como C, y quizás también debido a la potencia del interfaz gráfico de UNIX, las aplicaciones para clientes X son cada vez más solicitadas en todo tipo de entornos de trabajo.

gestor de ventanas mwm ("OSF/Motif Window Manager").

La elección de este gestor se debe principalmente al gran desarrollo que presenta respecto a otros, en todo lo que se relaciona con las librerías Motif que incorpora. Este gestor permite el incremento de la eficiencia en la programación de nuestras aplicaciones.

Por otro lado, Motif Windows Manager es un manejador de ventanas soportado en bastantes plataformas Unix, con lo que, una aplicación creada en nuestro Linux no precisa de ningún cambio para ser transportado, por ejemplo, a un HP-UX con el mismo gestor.

Figura 1.



El gestor de ventanas Motif: mwm

Si anteriormente decíamos que mwm era un intermediario entre los clientes y el servidor, debemos dejar claro que, a rasgos generales, el gestor de ventanas es un cliente más, pero con una peculiaridad especial. Mwm tiene la posibilidad de solicitar que todas las peticiones que reciba el servidor X de los demás clientes, relacionadas con la estructura de las ventanas (tamaño, posición, apariencia, etc), sean redirigidas con el propósito de que él tenga la posibilidad de tratarlas.

Debido a esta razón, sólo se permite tener un gestor de ventanas en proceso de ejecución.

Por lo tanto, este cliente tan especial, recibirá peticiones siempre y cuando una aplicación desee situar una ventana en nuestra pantalla, y será mwm quien se ecargará de decidir el aspecto que deberá presentar ésta.

Por otra parte, el gestor de ventanas permite al usuario la posibilidad de manipular la organización de las ventanas. Esta interacción, se lleva a cabo mediante la utilización del ratón.

El gestor de mwm es un cliente más, con la matización de que recibe todas las peticiones del servidor relacionadas con la estructura de las ventanas de Motif, siendo éste quien se encarga de tratarlas

Otra característica que posee nuestro gestor de ventanas es la incorporación de una serie de funciones que facilitan la ejecución de las tareas más habituales. Estas funciones serán tratadas más adelante. Después de instalar el paquete de OSF/Motif, para que mwm comience a funcionar bastará con añadir una línea en el archivo .xinitrc de nuestro directorio $HOME que contenga la palabra mwm. En este archivo se incluyen todas las instrucciones necesarias que deben ser ejecutadas, siempre que iniciemos el servidor X.

Escribiendo mwm en este archivo indicamos, que el gestor de ventanas que queremos que se ejecute por defecto en el arranque, sea Motif Windows Manager. Para otros sistemas operativos como el de HP-UX el archivo que se deberá modificar será .x11start.

Configuración de mwm

Siguiendo la filosofía Unix, mwm permite la reconfiguración de todos los recursos de los clientes mediante las modificaciones realizadas sobre una serie de ficheros ordinarios utilizando cualquier editor de textos; típicamente el vi. Entre todos los

ficheros de configuración de recursos generales, los que se consideran más importantes son los siguientes:

```
/usr/lib/X11/sys.Xdefaults
/usr/lib/X11/system.mwmrc
```

Para personalizar el entorno según el usuario, estos ficheros deberán ser copiados a su directorio HOME con los siguientes nombres:

```
$HOME/.Xdefaults
$HOME/.mwmrc
```

Pero, ¿cuál es el cometido de estos dos ficheros?.

- Configuración de recursos: .Xdefaults.

El fichero ".Xdefaults" es el archivo que se encarga de definir cada uno de los recursos pertenecientes a cada cliente específico.

La modificación del fichero .Xdefaults define los recursos de cada cliente, permitiendo la personalización de nuestras aplicaciones, tales como el color o la fuente de letra

De esta manera, las modificaciones de este fichero con nuestro editor de textos permitirán personalizar una serie de propiedades específicas de las aplicaciones tales como el color de fondo, fuente de letra, geometría de la ventana en la que se ejecuta, etc., además de los atributos específicos de cada cliente.

La sintaxis general en la que se encuentran los valores de los recursos en este fichero corresponde al patrón que se expone en este texto:

```
Nombre_cliente*recurso:      valor_recurso
```

De esta forma los recursos del cliente xclock podrían estar configurados como en el siguiente ejemplo:

```
XClock*background:    black
XClock*foreground:    magenta
XClock*hands:         orchid
XClock*highlight:     magenta
XClock*analog_        TRUE
```

Existe otra sintaxis específica que sirve para los patrones de los recursos pertenecientes a los subcomponentes de una aplicación:

```
Nombre_cliente*subcomponente*recurso:
    valor_recurso
```

Observamos esta sintaxis en el siguiente ejemplo sobre los subcomponentes del cliente XTerm:

```
XTerm*scrollBar*background: blue
XTerm*scrollBar*foreground: maroon
```

Utilizando el carácter comodín * podemos definir el valor de un recurso para varios clientes:

```
*background:                  green
XTerm*background:             white
XTerm*scrollBar*background:   black
```

Obviamente el valor que se obtiene con la sintaxis específica, tiene precedencia sobre el valor más general, que será reservado para todos aquellos clientes que dejen sin definir este recurso de forma explícita.

Este tipo de patrones para definir el valor de un recurso se almacenan en la base de datos existente en la memoria del servidor X y que se carga directamente desde el fichero .Xdefaults. Podemos especificar otros ficheros de configuración de recursos, que serán similares al .Xdefaults, y que pueden ser añadidos a la base de datos con la orden **xrdb**. Los ficheros de recursos deben ser cargados antes de arrancar la aplicación con la siguiente sintaxis:

```
xrdb -merge <Nombre_Fichero_de_Recursos>
```

Otra cualidad del mwm es la capacidad de poder declarar los recursos de la ventana raíz. La ventana raíz (fondo de la pantalla) actúa como ventana principal sobre la cual se sitúan las demás ventanas de las aplicaciones o clientes.

Así por ejemplo, para referirnos a la posición de una ventana, utilizamos las coordenadas de la ventana raíz, sobre la que se encuentra nuestra ventana.

En el archivo .Xdefaults también podemos declarar los recursos de la ventana principal.

Algunos de estos podrían ser los que se enumeran a continuación:

- Mwm*topShadowColor: black 'Sombreado izquierdo y superior de las ventanas inactivas.
- Mwm*bottomShadowColor: blue 'Sombreado derecho e inferior de las ventanas inactivas.
- Mwm*background: black 'Color de fondo para el borde de las ventanas.
- Mwm*keyboardFocusPolicy: explicit 'Señala que elemento indica cual es la ventana activa. El otro valor que puede tener es pointer.
- Mwm*iconPlacement: left bottom 'Situación de los iconos en pantalla.

También podemos aprovechar la ejecución del cliente xsetroot que permite configurar la ventana raíz con las siguientes opciones:

```
xsetroot -bitmap Nombre_bitmap -bg color -fg color
```

Valiéndonos de la opción -bitmap se dispone de la posibilidad de cambiar el fondo de la ventana raíz, utilizando para ello una imagen con el mismo formato mencionado anteriormente.

Por último indicaremos otra manera de especificar el valor de los recursos sin utilizar las especificaciones del archivo .Xdefaults: las opciones al cargar un cliente. La mayoría de los clientes ofrecen posibilidades que permiten configurar algunos de sus recursos cuando van a ser ejecutados desde la línea de comandos. Observemos las opciones de ejecución del cliente xterm con este ejemplo:

```
xterm  -bg black  -cr green -tittle Terminal -geo-
       metry 80x25
```

Al ejecutar esta línea tendremos un emulador de terminal con fondo negro, el cursor será verde, el título de la ventana será "Terminal" y nuestra ventana se visualizará a partir del píxel 80 de largo y 25 de alto de la ventana raíz.

Mwm permite la reconfiguración de los clientes mediante las modificaciones realizadas sobre ficheros ordinarios, utilizando un editor de textos

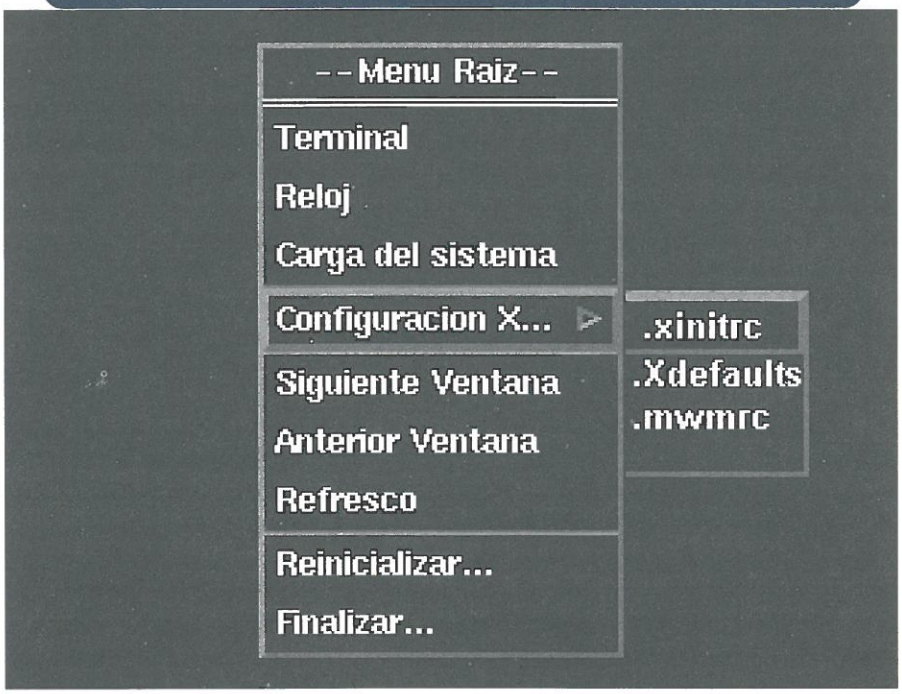
Para ver las opciones de los clientes siempre podemos acudir al manual en línea que proporciona Unix mediante el comando man <*cliente*>.

- **Configuración de menús: .mwmrc**

Como ocurre con la modificación del fichero de recursos, podemos modificar el fichero .mwmrc mediante un editor de textos. El fichero ".mwmrc" tiene asignadas varias funciones, siendo la primera de ellas la dedicada a la creación de menús.

La creación de menús ofrece una forma fácil de manipular el sistema mediante una interfaz visual normalmente accesible mediante el ratón o una combinación especial de teclas. La estructura de un menú en el fichero .mwmrc es la que se enumera:

Figura 2.




```
Menu Nombre_menú
{
      opción 1 [mnemónico] [acelerador] función
      opción 2 [mnemónico] [acelerador] función
        .          .          .          .
        .          .          .          .
      opción n [mnemónico] [acelerador] función
}
```

En primer lugar se asigna un nombre al menú, ya que todos tienen su nombre. Así el emergente en la ventana principal se llama RootMenu.

Opción se refiere al texto que permitirá identificar cada una de las distintas posibilidades del menú. El mnemónico y el acelerador se introducen si queremos que una pulsación de teclas active una determinada opción.

En cuanto a la función, se trata de un objeto que realiza tareas específicas. En primer lugar entre las funciones permitidas están las operaciones con los menús.

- f.menu <Nom_menú> asocia un menú con el submenú Nom_menú
- f.separator dibuja una línea entre las opciones del menú
- f.title escribe el título del menú en la posición asignada

La creación de menús proporciona una forma fácil de manipular el sistema mediante una interfaz visual accesible a través del ratón

Gestión de ventanas:

- f.circle_down coloca la ventana superior en el fondo de la pila.
- f.circle_up coloca la ventana inferior en el tope de la pila.
- f.lower coloca la ventana activa en el fondo de la pila.
- f.minimize minimiza la ventana representando su icono.

- f.maximice maximiza una ventana a su tamaño máximo.
- f.move activa el movimiento de una ventana.
- f.raise coloca la ventana activa en el tope de la pila.
- f.resize activa el redimensionamiento interactivo de una ventana.
- f.refresh redibuja todas las ventanas de nuevo.

Ejecución de programas:

- f.exec " cmd &" ejecuta el comando cmd usando /bin/bash
- f.restart reinicializa mwm
- f.kill elimina el cliente que crea una ventana
- f.nop no operar (no hace nada)

La función es un objeto que realiza una serie de operaciones específicas, entre ellas operaciones con menús

La estructura para crear el menú de la figura 1.2, sería la siguiente:

```
Menú RootMenu
{
    "—Menu Raiz—"        f.title
    "Terminal"           f.exec "xterm &"
    "Reloj"              f.exec "xclock &"
    "Carga del sistema"  f.exec "xload &"
    no-label             f.separator
    "Configuracion X..." f.menu
    M_Configuracion
    no-label             f.separator
    "Siguiente Ventana"  f.circle_up
    "Anterior Ventana"   f.circle_down
    "Refresco"           f.refresh
    no-label             f.separator
    "Reinicializar..."   f.restart
    "Finalizar..."       f.quit_mwm
}

! Submenu Configuracion
Menu M_Configuracion
{
    ".xinitrc"     f.exec "xterm -e vi
    $HOME/.xinitrc &"
    ".Xdefaults"   f.exec "xterm -e vi
    $HOME/.Xdefaults &"
    ".mwmrc"       f.exec "xterm -e vi
    $HOME/.mwmrc&"
}
```

Asociación de teclas y eventos de ratón: .mwmrc

La segunda función de .mwmrc consiste en definir las pulsaciones de combinaciones de teclas para realizar funciones de manera más rápida, tales como desplegar menús o ejecutar una de sus opciones. También es misión del .mwmrc declarar los eventos que se suceden tras la pulsación de alguno de los botones del ratón bajo un contexto definido.

La pulsación de una combinación de teclas puede significar la ejecución de una opción de un menú, en este caso hablaremos de aceleradores. La ventaja principal es que no hará falta desplegar el menú para ejecutar esa opción, con lo que se ganará tiempo. Los aceleradores también se definen en la declaración de menús dentro del archivo .mwmrc.

Si por el contrario tenemos desplegado un menú, también podremos ejecutar una opción asignándole un mnemónico. Un mnemónico no consiste más que en resaltar una tecla que simula el evento correspondiente a la pulsación del ratón sobre esa opción. Para dar una explicación práctica utilizaremos el menú de la figura 1.2. Al definir este menú podríamos haber incluido mnemónicos y aceleradores para las opciones. Pongamos el siguiente ejemplo de ejecución del terminal xterm:

```
Menú RootMenu
{
"—Menu Raiz—"                    f.title
"Terminal" _T   Alt<esc|<Key>|F5   f.exec
      "xterm &"
.
.
.
}
```

Al definir de está forma el menú, con la pulsación de la tecla T, ejecutamos nuestro terminal si tenemos desplegado el menú gracias al mnemónico _T.

Si no estuviese desplegado el menú, el acelerador "Alt<esc|<Key>|F5", también permite la ejecución de esta opción mediante la pulsación de la combinación de teclas Alt+F5.

También en el archivo .mwmrc podemos definir qué debe ocurrir si se produce el evento click en cualquier parte de la ventana raíz. Esta declaración se produce en la sección *Buttons DefaultButtonBindings*. Veamos un ejemplo con esta característica:

```
Buttons DefaultButtonBindings
{
<Btn1Down>    root          f.menu RootMenu
<Btn3Down>    root          f.menu RootMenu
<Btn1Down>    frame|icon    f.raise
}
```

Con las dos primeras líneas de esta sección, especificamos que cuando sea pulsado el botón 1 ó 3 del ratón sobre la ventana raíz se visualice el menú raíz. En la tercera línea asociamos la función f.raise (colocar la ventana la primera de la pila) a la pulsación del botón 1 del ratón sobre cualquier ventana de una aplicación o cliente.

Conclusión

Una vez aplicados estos conocimientos habremos llegado a configurar todas nuestras aplicaciones y clientes a nuestro estilo, sin ninguna dificultad, aprovechando la potencia del entorno gráfico del sistema operativo Linux y su eficiente gestor de ventanas mwm.

Servicio de correo electrónico

Carlos Álvaro González (sympha@hotmail.com)

El servicio de transferencia de correo electrónico entre máquinas pertenecientes a Internet es de los más antiguos que se vienen ofreciendo en la historia de esta gran red. También es, quizás, el más representativo de la filosofía cliente servidor dentro de la extensa familia de protocolos TCP/IP.

El servicio de correo electrónico permite llevar a cabo el envío y recepción de mensajes con usuarios de otros ordenadores de la red. El modo más sencillo de implementar este servicio es mantener ficheros de correo, llamados buzones, en cada uno de los sistemas en los que se pretende intercambiar mensajes con alguno de sus usuarios.

Este sistema se basa en un método que permite añadir un mensaje a esos ficheros, o bien recuperarlo. El problema principal aparece cuando los ordenadores entre los que se intercambia correo no tienen continuamente activo el gestor de correo (bien por estar apagados o por no permitir la multitarea).

Esto se solventa mediante la instalación de mail servers, sobre los que hablaremos más adelante. Por el momento, centrémonos en la gestión y distribución clásicas de este tipo de mensajería.

Cliente, servicio, servidor y protocolo

El cliente e-mail es un programa habitualmente llamado "mail" que permite la lectura de los ficheros buzón a cada usuario autorizado. También es capaz de manejar el "spool" de correo, es decir, de enviar mensajes que serán leídos por el servidor e-mail para poder ser distribuidos a otros usuarios.

El servicio encargado de proporcionar el intercambio de correo electrónico entre las máquinas se denomina servicio de e-mail, y puede ser ofrecido por distintos programas daemon, de los que el más extendido se llama "sendmail".

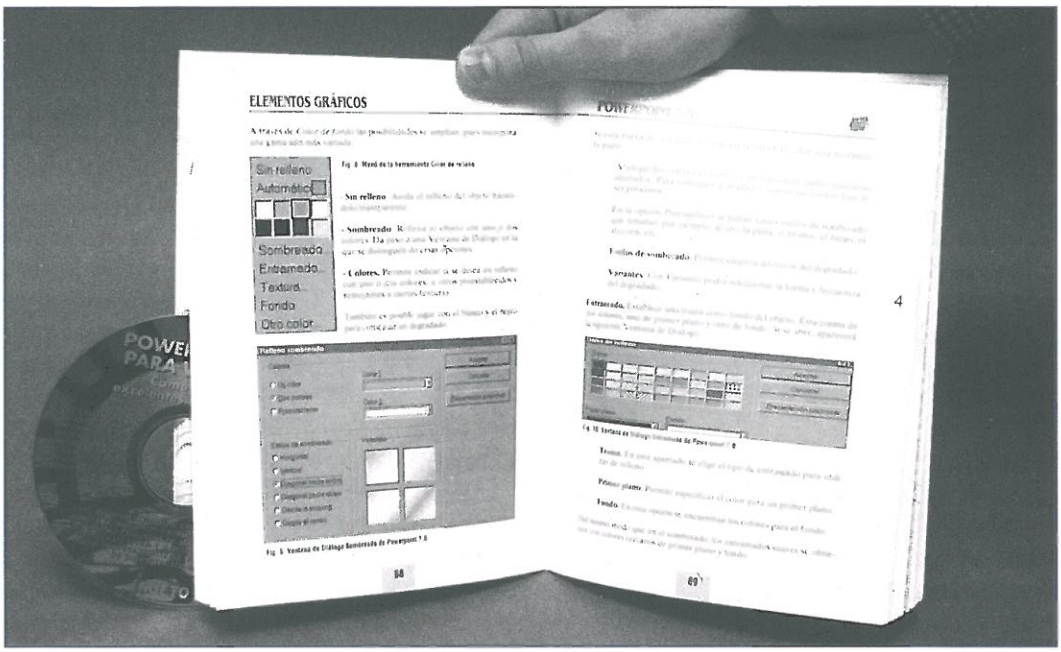
El daemon sendmail es el servidor de correo electrónico más popular entre las máquinas de Internet. Según algunos estudios, constituye entre el setenta y el ochenta por ciento de las implementaciones existentes hoy en día.

El programa más extendido para el intercambio de correo electrónico se denomina sendmail

También se conoce con el nombre de "delivery", o bien distribuidor de correo. Realiza su función manteniéndose a la escucha del socket 25, comunicándose con los daemons de otros sistemas para recibir el correo entrante y enviar el correo saliente.

La mayoría de anuncios editoriales te muestran los libros cerrados

NOSOTROS HEMOS PENSADO QUE PREFIERES VERLOS ABIERTOS



Libro + CD-ROM

por sólo

1.995

ptas.

IVA incluido

- ✓Explicaciones paso a paso
- ✓Ejemplos prácticos a todo color
- ✓Imágenes de cada una de las pantallas que encontrarás en tu ordenador

Biblioteca de Informática AVANZADA

Títulos que componen la colección "Biblioteca Informática Avanzada":

- EXPLORANDO INTERNET
- 3 D STUDIO V.4
- AUTOCAD 13
- MÚSICA DIGITAL POR ORDENADOR
- REALIDAD VIRTUAL
- LENGUAJES HTML, JAVA Y CGI
- WORD PARA WINDOWS 95
- COREL 6.0
- VISUAL BASIC 4.0
- POWERPOINT 7.0 PARA WINDOWS 95
- EXCEL 7.0 PARA WINDOWS 95
- JAVA. EL LENGUAJE DE INTERNET
- MICROSOFT WORD 97
- MICROSOFT ACCESS 97
- MICROSOFT EXCEL 97

Distribuidores autorizados en librerías

COMUNIDAD DE MADRID / CASTILLA LA MANCHA		**CANARIAS**	
DISTRIFORMA S.A.	Tfno. 91 - 501 4749	GARCIA PRIETO LIBROS S.L.	Tfno. 922-820026
CATALUÑA		**BALEARES**	
MIDAC LLIBRES S.L.	Tf. 93-421 18 95	PONENT LLIBRES S.L.	Tfno. 971-430339
ANDALUCIA OCCIDENTAL/ EXTREMADURA		**VALENCIA - CASTELLÓN**	
DISTRIBUCIÓN DE EDICIONES RDGUEZ. SANTOS S.L.	Tfno. 95 - 418 04 75	ADONAY S.L.	Tfno. 96-3793151
ANDALUCIA ORIENTAL		**ALICANTE - MURCIA - ALBACETE**	
DISTRIBUCIONES DEL MEDIODIA S.A. (ZÓCALO)	Tfno. 958-550278	LA TIERRA LIBROS S.L.	Tfno. 96-5110192
CASTILLA - LEÓN		**PAÍS VASCO - NAVARRA**	
ARCADIA S.L.	Tfno. 983-395049	YOAR S.L.	Tfno. 948-302239
GALICIA		**ARAGÓN - LA RIOJA**	
LUIS REY ABELLA (DISGALIBRO)	Tfno. 981-795754	ICARO DISTRIBUIDORA S.L.	Tfno. 976-126333
ASTURIAS - CANTABRIA			
DISTRIBUCIONES CIMADEVILLA S.A.	Tfno. 98-5167930		

ABETO

c/ Aragoneses, 7. 28.108 Alcobendas (MADRID)
Tel.: (91) 661 42 11* Fax: (91) 661 43 86

En cuanto a la aplicación TCP/IP, se utiliza el protocolo SMTP (Simple Mail Transfer Protocol), el cual se caracteriza por su eficiencia, sencillez y facilidad de depuración, gracias a los mensajes que acompañan a sus comandos.

Funcionamiento detallado de SMTP

En este artículo se va a mostrar al lector, de forma detallada, cómo funciona una conexión SMTP. Es un ejemplo claro del "modo de hacer" de los protocolos Internet que podemos reproducir fácilmente en las máquinas de nuestra intranet. Utilizaremos para ello dos de las máquinas de nuestra intranet, llamadas 'natacha' y 'sympha'.

Aunque 'natacha' es un sistema Unix distinto de Linux y 'sympha' es un servidor OS/2, el mismo ejemplo, utilizando 'menteuse' y 'andora', ambas con Linux, sería sumamente similar y apenas precisaría de cambios.

La gestión de todo lo concerniente a la red por parte de IP y TCP deja a los programas de aplicación como un simple intercambio de comandos y datos. Por ejemplo, el protocolo de correo, del que podemos hacer un seguimiento muy sencillo, trabaja de la manera que vamos a describir a continuación.

Nuestro programa de mail abre una conexión contra el mail server remoto. Entonces, envía su nombre de máquina local, así como el nombre del emisor, el buzón de destino y un comando diciendo que empieza el texto del mensaje.

En este punto, el servidor finaliza el tratamiento de lo que ha asumido como comandos y comienza a aceptar el mensaje hasta que recibe una marca especial (sencillamente, un punto como principio de línea). Después de esto, ambos programas entienden que el envío de comandos ha sido retomado.

Para ver cómo es el funcionamiento, vamos a suponer que una computadora llamada natacha.intranat quiere enviar el siguiente mensaje:

```
Date: Sat, 22 Jun 96 04:22:28 GMT
From: charlie@natacha.intranat
To: root@sympha.intranat
Subject: dommage

J'ai égaré un fichier.
```

El formato del mensaje está basado en Internet standard (RFC 822), donde también se especifica que el mensaje debe transmitirse como net ASCII (ASCII con <cr><lf>), con unas líneas de cabecera determinadas, una línea en blanco y, a continuación, el cuerpo del mensaje que deseamos enviar.

Los formatos estándares, diseñados para las direcciones, se han hecho mucho más flexibles

Las direcciones se indican de la forma clásica por todos conocida, como "usuario@máquina".

Éste era el formato que fue diseñado inicialmente, pero rápidamente los estándares se hicieron mucho más flexibles para contemplar casos de máquinas integradas bajo dominios, con la nomenclatura "usuario@máquina.subdominio(s).dominio", donde puede que una de las máquinas sea el servidor central de correo de las demás.

Por lo demás, también existe la posibilidad de que el campo "usuario" no sea realmente un usuario del sistema. Esto permite manejar listas de dichos usuarios y nombres genéricos. En el ejemplo que estamos desarrollando, la primera operación consiste en preguntar quién maneja el correo para sympha.intranat.

El servidor de correo responde que sympha.intranat maneja su propio correo. El programa local de correo en natacha consulta la dirección de sympha.intranat, (al servidor de nombres, si lo hubiera), y obtiene su dirección Internet. Entonces, abre una conexión al socket 25 (servidor SMTP) y tiene comienzo la conversación deseada.

El intercambio de mensajería, en lo que se refiere a la aplicación, será el siguiente que se enumera:

```
(sympha)  220 SYMPHA.INTRANAT SMTP
      Service at 22 Jun 96 05:02:31 GMT
(natacha) HELO natacha.intranat
(sympha)  250 SYMPHA.INTRANAT - Hello,
      NATACHA.INTRANAT
(natacha) MAIL From:<charlie@natacha.intra-
      nat>
(sympha)  250 MAIL accepted
(natacha) RCPT To:<root@sympha.intranat>
(sympha)  250 Recipient accepted
(natacha) DATA
(sympha)  354 Start mail input; end with
      <CRLF>.<CRLF>
(natacha) Date: Sat, 22 Jun 96 04:22:28 GMT
(natacha) From: charlie@natacha.intranat
(natacha) To: root@sympha.intranat
(natacha) Subject: dommage
(natacha)
(natacha) J'ai égaré un fichier.
(natacha) .
(sympha)  250 OK
(natacha) QUIT
(sympha)  221 SYMPHA.INTRANAT Service clo-
      sing transmission channel
```

Para poder reproducir todo esto, por ejemplo por medio de menteuse y andora, podemos suplantar al programa de mail de menteuse haciendo un telnet a andora, pero al puerto 25, escribiendo la orden siguiente:

```
telnet andora 25
```

A partir del momento en hayamos escrito esta imprescindible línea, para andora nosotros ya nos habremos convertido en un programa cliente de correo

que ha solicitado una conexión, y como tal nos va a tratar. Continuemos con nuestra explicación.

Los comandos utilizan texto normal, característica del estándar Internet, lo que resultará muy útil a la hora de ver qué es lo que está sucediendo y ser capaces de diagnosticar problemas.

Por ejemplo, el programa de mail mantiene un log de cada conversación. Si algo sale mal, este log puede ser enviado al administrador de correo, quien tiene la posibilidad de interactuar directamente contra el servidor SMTP.

En protocolos que sean un poco más complicados, este proceso no es práctico en absoluto y se utilizan formatos binarios. Sin embargo, suelen estructurarse como registros de C (struct) o bien de Pascal (récord).

Otra característica del ámbito Internet es que las respuestas empiezan con un número definido en el protocolo, lo que evita ambigüedades.

El log del programa de mail, puede enviarse al administrador que interactua con el servidor SMTP

Este número es lo que realmente tiene efecto en la operación de los programas; el texto en realidad es meramente informativo. De entre estos números de respuesta, lo verdaderamente importante es el primer dígito:

Un número de respuesta que empieza por 2 indica una ejecución satisfactoria.

Los que empiezan por 3 requieren una acción a continuación. El rango de los que comienzan por 4 es de error temporal (por ejemplo, cuando el disco se encuentra lleno), y el mensaje tendrá que ser salvado para reintentarlo más tarde. El 5 es error permanente, (por ejemplo, no existe recipiente), y el mensaje tendrá que ser devuelto al emisor con un código de error.

Cada sesión empieza con un HELO, que proporciona el nombre del sistema que ha iniciado la conexión. Después de eso, por medio de una sentencia MAIL será especificado el emisor del correo y un grupo de sentencia RCPT de los recipientes de destino.

Finalmente, se envían los datos mediante DATA, acabándolos con una línea que contiene tan sólo un punto en su primera columna.

Si hubiese que incluir una línea de esas características como parte del mensaje, entonces habría que hacer que el punto fuera doble.

Después de que el mensaje haya sido aceptado por el receptor, el programa de mail tiene dos posibilidades: enviar otro mensaje o bien terminar la conversación mediante QUIT.

El cliente mail: Recepción de correo

Ahora nos vamos a centrar en algunas de las posibilidades más interesantes que nos ofrece el programa cliente mail si nos encontramos bajo Linux. Como se comentó anteriormente, la llamada a 'mail' sin argumentos permitirá a un usuario leer el correo que se encuentre almacenado en su buzón.

Por defecto, los ficheros buzón de correo se almacenan en el directorio /usr/spool/mail con el login name de cada usuario. Una vez que los mensajes han sido leídos, desaparecen del buzón y se almacenan en el fichero 'mbox', que se encuentra en el directorio de conexión del usuario. Para leer el fichero mbox, en vez del buzón correspondiente, se puede utilizar la opción -f.

HELO proporciona el nombre del sistema que ha iniciado la conexión y después con la sentencia MAIL se especifica el emisor de correo y un grupo RCPT

La lectura de un buzón de correo presenta los mensajes perfectamente ordenados, cada uno con su número de mensaje, con objeto de poder ser accesibles de manera independiente.

Una vez seleccionado un mensaje se puede disponer del mismo para llevar a cabo las siguientes funciones:

1. Visualizar su contenido mediante 'p' (print) o, simplemente, pulsando <intro>.
2. Borrarlo pulsando 'd' (delete). Esta acción es reversible mediante el comando 'u' (undo).
3. Responderlo, es decir, mandar correo al emisor del mensaje, con el comando 'r' (reply).

Para poder salir de nuestra sesión con mail se utiliza la orden 'q' (quit) o bien la orden 'x' (exit).

Esta última aborta la sesión con todos los cambios que hayamos realizado. Es decir, no efectúa nunca modificaciones sobre el propio buzón del usuario.

Lo explicado hasta ahora constituye lo necesario para una sesión habitual con 'mail'. Sin embargo, el programa 'mail' es bastante más completo.

Podemos obtener un manual detallado de sus posibilidades y funcionamiento mediante el comando 'man' de Linux, escribiendo "man mail". (En algunos sistemas, la ayuda para mail se obtiene escribiendo: "man mailx" o "help mail").

El cliente mail: Envío de correo

Si queremos enviar un mensaje a uno o más usuarios que se encuentren en nuestra red, basta con utilizar el comando 'mail' con los nombres de dichos usuarios como parámetros.

Se pueden dar algunas causas por las que un determinado mensaje no pueda ser distribuido, (por ejemplo, por un repentino error del sistema, o bien que el recipiente para determinado usuario no exista, etcétera).

Para enviar un mensaje el parámetro utilizado es el nombre del propio usuario

En ese caso, el mensaje que haya sido escrito no llegará a perderse, sino que será almacenado dentro de un fichero llamado "dead.letter", que se encuentra situado en el interior del directorio HOME del usuario.

Durante el proceso de escritura de los mensajes que se quieren enviar, se puede utilizar el carácter de escape tilde (~) al principio de una línea, para hacer una llamada a alguna de las funciones especiales que se encuentran a nuestra disposición. Si el juego de caracteres que estamos utilizando es el que se corresponde con el mapa "es.map", dentro de la consola de Linux el carácter tilde se puede obtener pulsando las teclas AltGr-4.

Nuevamente podemos obtener una referencia completa de las funciones disponibles consultando la ayuda de Linux.

Algunas de las que pueden ser más útiles para nuestros fines son las que mostramos a continuación:

- ~!comando - Ejecuta el comando indicado y retorna a la edición de correo.
- ~e - Invoca a editor de textos establecido. Esta función puede ser muy útil cuando estemos manejando mensajes que sean especialmente extensos.
- ~h - Edita para su modificación la información de cabecera del mensaje.
- ~q - Aborta el envío del mensaje, copiando lo que se ha escrito hasta ese momento en el fichero "dead.letter". (Debe estar activa la opción 'save').
- ~d - Recupera el contenido del fichero "dead.letter" añadiéndolo al mensaje en curso.
- ~wnomfich - Escribe el mensaje en el fichero "nomfich" y continúa con la edición.

Configuración del cliente mail

La activación y la desactivación de todas las opciones de comportamiento que posee el programa 'mail' se pueden realizar a través de dos comandos, denimonados "set" y "unset".

De todas maneras, no todo queda ahú. Se dispone, en efecto, de una segunda posibilidad que puede ser muy útil: también pueden establecerse si utilizamos convenientemente los ficheros de configuración.

En la configuración del cliente mail de Linux intervienen dos ficheros:

/etc/mail.rc
$HOME/.mailrc

/etc/mail.rc contiene una serie de opciones del cliente mail que pueden ser controladas mediante los comandos 'set' y 'unset'. De esta forma, se pueden activar y desactivar las distintas opciones booleanas.

Otras opciones se definen con variables que pueden adoptar distintos valores. La información completa sobre la composición del fichero 'mail.rc' se puede obtener tecleando el comando 'man mailx'.

La activación y desactivación de las opciones de comportamiento se realiza con los comandos "set" y "unset"

Las opciones de comportamiento del cliente mail pueden ser establecidas particularmente para cada usuario a través del fichero '.mailrc', que se encuentra en los distintos directorios de conexión.

Conclusión

Tradicionalmente, Simple Mail Transfer Protocol ha sido el protocolo de correo utilizado entre máquinas de Internet. Sin embargo, para la distribución de correo por SMTP las máquinas implicadas deben estar siempre en funcionamiento.

Si esto no se produce, será necesaria entonces la intervención de otro subsistema de correo; el Post Office Protocol (POP), del que comentaremos algunos aspectos en un artículo próximo.

Control del módem con Visual Basic

Constantino Sánchez Ballesteros (constantino@redestb.es)

El control *MSComm* proporciona una interfaz con un conjunto estándar de comandos de comunicaciones. Permite establecer una conexión con un puerto serie, conectar con otro dispositivo de comunicaciones (por ejemplo, un módem), emitir comandos, intercambiar datos y supervisar y responder a varios eventos y errores que pueden producirse durante una conexión serie.

El control *MSComm* proporciona dos formas diferentes para el tratamiento de las comunicaciones:

- Las comunicaciones controladas por eventos son un método muy poderoso para el tratamiento de interacciones con el puerto serie.

  En muchas situaciones, es necesario saber el momento en que tiene lugar un evento, por ejemplo, cuándo llega un carácter o cuándo se produce un cambio en las líneas de Detección de portadora (CD) o Petición de envío (RTS).

  En tales casos, se utiliza el evento *OnComm* del control *MSComm* para interceptar y tratar estos eventos de comunicaciones. El evento *OnComm* también detecta y trata los errores en las comunicaciones.

- También puede sondear los eventos y errores comprobando el valor de la propiedad *CommEvent* después de cada función crítica de su programa. Esta alternativa es preferible si la aplicación es pequeña y autónoma. Por ejemplo, si estamos escribiendo un marcador de llamadas telefónicas sencillo, no tiene sentido generar un evento después de recibir cada carácter, ya que los únicos caracteres que pensamos recibir son las respuestas de aceptación que enviadas por el módem.

Cada uno de los controles *MSComm* que utilicemos se corresponde con un puerto serie. Si necesitamos tener acceso a más de un puerto serie en nuestra aplicación, debemos usar más de un control *MSComm*. La dirección del puerto y la dirección de la interrupción se pueden cambiar desde el Panel de control de Windows.

Con el control MSComm podemos crear desde marcadores de teléfono hasta terminales para la conexión entre equipos

Aunque el control *MSComm* tiene muchas propiedades importantes, hay algunas con las que es necesario que el lector se familiarice.

El control MSComm de Visual Basic 5.0 permite ofrecer a nuestras aplicaciones una funcionalidad sencilla para manejar los puertos series o una avanzada para crear una herramienta de comunicaciones completa, controlada mediante eventos.

Propiedades y descripción

Entre las más importantes se pueden citar las siguientes propiedades, :

- *CommPort.* Establece y devuelve el número del puerto de comunicaciones.
- *Settings.* Establece y devuelve la velocidad en baudios, paridad, bits de datos y bits de parada en forma de cadena.
- *PortOpen.* Establece y devuelve el estado de un puerto de comunicaciones. También se encarga de abrir y cierra un puerto.
- *Input.* Devuelve y quita caracteres del *buffer* de recepción.
- *Output.* Es la propiedad que se encarga de escribir una cadena de caracteres en el *buffer* de transmisión.

Entre las posibles aplicaciones que podemos generar con *MSComm* destacan las enumeradas a continuación:

El marcador de números de teléfono, la supervisión de la llegada de datos y la creación de un programa completo de terminal.

Fundamentos de la comunicación serie

Todos los ordenadores se suministran con uno o más puertos serie, que se denominan sucesivamente COM1, COM2, etc. En un equipo estándar, normalmente el *mouse* (ratón) estará conectado al puerto COM1. En el puerto COM2 es posible que se encuentre un un módem conectado, en COM3 un escáner, etc.

La función esencial del puerto serie es actuar como intérprete entre la CPU y el dispositivo serie. Al enviar datos desde la CPU a través del puerto serie, los valores de tipo *byte* se convierten en series de *bits*. Cuando se reciben datos, se realiza la operación complementaria, convirtiéndose los bits recibidos en valores de tipo *byte*.

Para completar la transmisión de los datos es necesario otro nivel de interpretación. En la parte del sistema operativo, Windows utiliza un controlador de comunicaciones, **Comm.drv**, que permite enviar y recibir datos mediante las funciones estándar de la API.

El fabricante del dispositivo serie proporciona un controlador que conecta este hardware con Windows. Cuando utilizamos el control *MSComm*, estamos ejecutando funciones de la API que interpreta el controlador **Comm.drv** y que se transfieren al controlador del dispositivo.

Como programadores, sólo debemos preocuparnos de cómo se efectúa la interacción, sin prestar atención a los detalles, de los cuales se encargará el propio Widows. En este caso concreto, mediante la programación con Visual Basic sólo tenemos que preocuparnos de la interfaz que proporciona el control *MSComm* para acceder a las funciones de la API del controlador de comunicaciones de Windows. Es decir, sólo necesitamos establecer y supervisar las propiedades y eventos del control *MSComm*.

Establecimiento de la conexión serie

El primer paso para utilizar el control *MSComm* consiste en establecer la conexión con el puerto serie. A continuación se detallan las propiedades necesarias para establecer la conexión:

- **CommPort, Settings, y PortOpen**

Procederemos con éstas a la apertura del puerto serie del siguiente modo:

```
MSComm1.CommPort = 2
MSComm1.Settings = "9600,N,8,1"
MSComm1.PortOpen = True
```

La propiedad *CommPort* determina el puerto serie que se va a abrir. En el ejemplo anterior, se establece el valor a 2 (COM2) y se conecta con el módem (que suponemos en dicho puerto). Podemos establecer la propiedad *CommPort* a cualquier número entre 1 y 16 (el valor por defecto es 1).

Sin embargo, si asignamos este valor a un puerto COM que no existe en el sistema en el que se ejecuta la aplicación se producirá un error.

El programador únicamente debe preocuparse de efectuar la interacción, sin necesidad de prestar atención a los detalles ya que de éstos se ocupará el propio sistema operativo Windows

La propiedad *Settings* permite especificar la velocidad en baudios, la paridad y el número de bits de datos y parada. De forma predeterminada, la velocidad en baudios es 9600. La paridad sirve para la validación de los datos. Normalmente, no llega a utilizarse y se establece a "N". El valor de *bits* de datos indica el número de *bits* que representa un bloque de datos. El *bit* de parada indica cuándo se ha recibido un bloque de datos.

Después de especificar el puerto que se va a abrir y la forma en que se realizará la comunicación de los datos, para establecer la conexión podemos utilizar la propiedad *PortOpen*, que es un valor booleano, *True* o *False*. Sin embargo, si el puerto no funciona, si la propiedad

CommPort no se ha establecido correctamente o si el dispositivo no admite la configuración especificada, se producirá un error o puede que el dispositivo externo no funcione correctamente. Si establecemos la propiedad *PortOpen* a *False*, se cierra el puerto.

Trabajo con un módem

En la mayoría de las ocasiones, utilizaremos el control *MSComm* que sirve para programar la aplicación de forma que funcione con un módem.

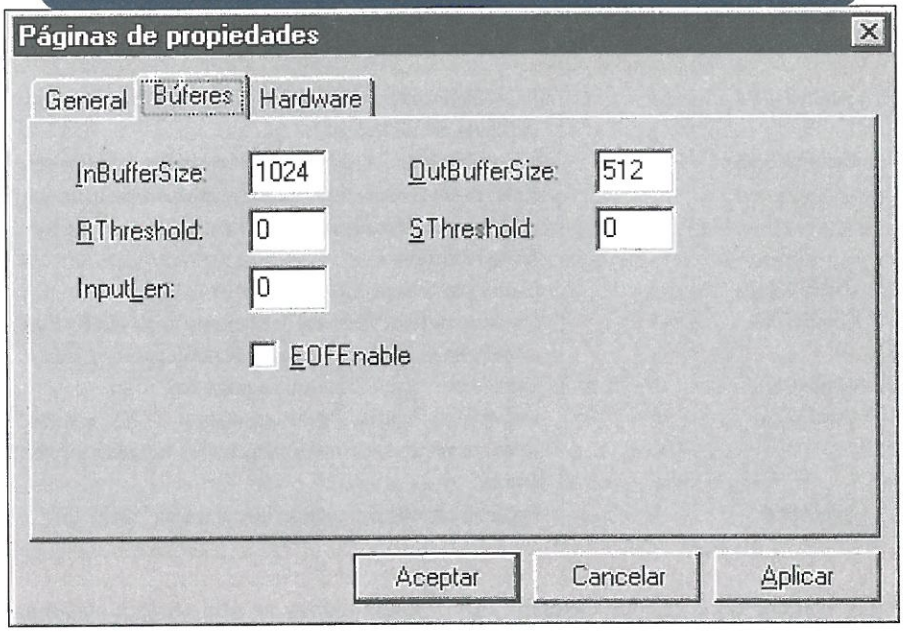
Valiéndose del control *MSComm* tendremos la posibilidad de utilizar el conjunto de comandos estándar. Este control compatible con **Hayes,** sirve para marcar un número de teléfono o con la finalidad de conectarse e interactuar con otro módem.

Entre las propiedades para el establecimiento de la conexión figura Setting, que permite especificar la velocidad en baudios, la paridad y el número de bits de datos y parada

Una vez que establezcamos la conexión con el puerto serie mediante las propiedades *CommPort*, *Settings* y *PortOpen*, utilizaremos la propiedad *Output* para activar el módem e interactuar con él.

La propiedad *Output* se utiliza para emitir los comandos que controlan la interacción entre dos modems. Por ejemplo:

Figura 1.



Activa el módem y marca un número determinado de teléfono.

```
MSComm1.Output = "ATDT 943-555523" & vbCr
```

En este ejemplo, el comando "AT" inicia la conexión, "D" marca el número y "T" especifica que el marcado es por tonos (para el marcado por pulsos se utilizaría "P"). Debemos incluir un carácter de retorno de carro (**vbCr**) cuando utilicemos un comando AT.

Cuando un comando se procesa con éxito, se obtendrá el código de resultado "*OK*". Podemos comprobar este código para determinar si el comando se ha ejecutado correctamente.

Establecimiento de las propiedades de los buffers

En el momento en que se abre un puerto, se crean *buffers* de transmisión y de recepción. Con el propósito de administrarlos, el control *MSComm* proporciona una serie de propiedades, que podemos establecer en tiempo de diseño a través de las páginas de propiedades del mismo.

- **Asignación de memoria de buffer**

En las propiedades *InBufferSize* y *OutBufferSize* especificamos la cantidad de memoria asignada a los *buffers* de recepción y transmisión. Los valores establecidos de manera predeterminada son los mostrados en la imagen anterior. Cuanto mayor sea el valor especificado, menos memoria habrá disponible para la aplicación. Sin embargo, si el *buffer* es demasiado pequeño, corremos el riesgo de desbordarlo, a menos que gestionemos la conexión con un determinado protocolo.

- **Las propiedades RThreshold y SThreshold**

Las propiedades *RThreshold* y *SThreshold* establecen o devuelven el número de caracteres recibidos en los *buffers* de recepción y transmisión antes de producirse el evento *OnComm*. El evento *OnComm* se utiliza para supervisar y responder a los cambios en el estado de la comunicación. Si se establecen ambas propiedades a 0, se impide que se produzca dicho evento. Si se asigna un

Tabla 1.

Constante	Valor	Descripción
comEvSend	1	En el *buffer* de transmisión hay menos caracteres que los indicados en *SThreshold*.
ComEvReceive	2	Se ha recibido el número de caracteres indicado en *RThreshold*. Este evento se genera continuamente hasta que utilicemos la propiedad *Input* para quitar los datos del buffer de recepción.
comEvCTS	3	Cambio en la línea "Listo para emitir" (CTS).
ComEvDSR	4	Cambio en la línea "Terminal de datos preparado" (DSR). Este evento sólo se produce cuando la línea DSR pasa de 1 a 0.
comEvCD	5	Cambio en la línea "Detector de portadora" (CD).
ComEvRing	6	Detectada la llamada. Puede que algunas UART (transmisores-receptores universales asíncronos) no admitan este evento.
comEvEOF	7	Recibido carácter de fin de archivo (carácter ASCII 26).

valor distinto de 0 (1, por ejemplo), el evento *OnComm* se producirá cada vez que se reciba un único carácter en cualquiera de los dos *buffers*.

- Las propiedades InputLen y EOFEnable

Estableciendo la propiedad *InputLen* a 0, el control *MSComm* leerá todo el contenido del *buffer* de recepción cuando se utilice la propiedad *Input*.

En el momento en que leamos los datos de un equipo cuyo resultado tiene formato de bloques de datos de longitud fija, podremos establecer el valor de esta propiedad como corresponda.

La propiedad *EOFEnable* se utiliza para indicar que se ha encontrado un carácter de final de archivo (EOF) durante la entrada de datos. Si se establece a *True*, la entrada de datos se detiene y se desencadena el evento *OnComm* para informarnos de que se ha producido esta situación.

- Administración de los buffers de recepción y transmisión

Como se ha indicado anteriormente, los *buffers* de recepción y transmisión se crean siempre que se abre un puerto. Estos buffers se utilizan para almacenar los datos de entrada y para transmitir los datos de salida. El control *MSComm* permite administrarlos a través de diversas propiedades con las que puede colocar y recuperar datos, obtener el tamaño de cada *buffer* y tratar datos de texto y binarios. La correcta administración de estos *buffers* es una parte importante del uso del control *MSComm*.

- El buffer de recepción

Mediante la propiedad *Input*, podemos almacenar datos en el *buffer* de recepción y también obtenerlos de él. Por ejemplo, si lo que pretendemos es recuperar datos del *buffer* de recepción y mostrarlos en un cuadro de texto, utilizaremos el código siguiente:

```
TxtDisplay.Text = MSComm1.Input
```

Sin embargo, para recuperar todo el contenido del *buffer* de recepción, primero se debe establecer la propiedad *InputLen* a 0. Esta operación puede llevarse a cabo bien en tiempo de diseño o en tiempo de ejecución.

Por otro lado podemos recibir los datos de entrada como texto o como datos binarios, si establecemos la propiedad *InputMode* a una de las siguientes constantes de Visual Basic. Éstas propiedades son: *comInputModeText* o bien *comInputModeBinary*. Así, recuperaremos los datos con formato de cadena o como datos binarios en una matriz de *bytes*. *comInputModeText* se utiliza para los datos que empleen el juego de caracteres ANSI y *comInputModeBinary* para todos los demás, como aquéllos que incluyan caracteres de control incrustados, caracteres nulos, etc.

En el momento en que se recibe cada *byte* de datos, se traslada al *buffer* de recepción y la propiedad *InBufferCount* se incrementa en una unidad.

Por tanto, es posible utilizar la propiedad *InBufferCount* para obtener el número de *bytes* del *buffer* de recepción. También podemos borrar el contenido del *buffer* de recepción si establecemos el valor de esta propiedad a 0.

- El buffer de transmisión

Mediante la propiedad *Output* , podemos enviar comandos y datos al *buffer* de transmisión.

Al igual que con la propiedad *Input*, podemos transmitir los datos como texto o como datos binarios. Sin embargo, la propiedad *Output* debe transmitir el texto o los datos binarios mediante la especificación de una cadena o de un tipo *Variant* de matriz de *bytes*.

Con la propiedad *Output*, podemos enviar comandos, cadenas de texto o datos de una matriz de *bytes*. Por ejemplo:

```
Envía un comando AT

MSComm1.Output = "ATDT 944-202020" & vbCr

' Envía una cadena de texto
MsComm1.Output = "Esto es una cadena de
    texto" & vbCr

' Envía datos de una matriz de bytes
MSComm1.Output = Out
```

Como se mencionó anteriormente, en este artículo las líneas de transmisión

deben terminar, con un carácter de retorno de carro (**vbCr**).

En el último ejemplo, que se expone, *Out* es de un tipo *Variant* definido como matriz de *bytes*:

```
Dim Out() As Byte
```

Si fuera un tipo *Variant* de cadena, se definiría de este modo:

```
Dim Out() As String
```

Podemos controlar el número de *bytes* del *buffer* de transmisión mediante la propiedad *OutBufferCount*.

Mediante la propiedad de Output, podremos enviar comandos, cadenas de textos o datos de una cadena de bytes

Por otro lado, también podemos borrar el contenido del *buffer* de transmisión si establecemos esta propiedad a 0.

Nota: *El control* **MSComm** *no puede enviar ni recibir datos binarios del juego de caracteres de doble byte (DBCS).*

Protocolo

Una de las tareas de la administración de los *buffers* de recepción y transmisión es asegurarse que el tráfico de datos tenga éxito; por ejemplo, que la velocidad con la que se reciben los datos no desborde los límites del *buffer*.

El término *protocolo* hace referencia al control de las comunicaciones, gestionando las transferencias del puerto hardware al *buffer* de recepción. Cuando llega un carácter de datos al puerto serie, el dispositivo de comunicaciones tiene que llevarlo al *buffer* de recepción para que el programa pueda leerlo.

Un protocolo de comunicaciones asegura que los datos no se perderán por un desbordamiento del *buffer*, lo que puede ocurrir cuando los datos llegan al puerto tan rápido es que el dispositivo de comunicaciones no puede llevarlos al *buffer* de recepción.

Podemos establecer la propiedad *Handshaking* con la finalidad de especificar el protocolo de comunicación, que va a utilizar la aplicación. De forma predeterminada, está establecida a *comNone* (ninguno). Sin embargo, se dispone de la posibilidad de especificar cualquiera de los protocolos siguientes:

Opción	Valor	Descripción
comNone	0	Sin protocolo (predeterminado).
comXOnXOff	1	Protocolo XON/XOFF.
comRTS	2	Protocolo RTS/CTS (Petición para emitir/Listo para emitir).
comRTSXOnXOff	3	Ambos protocolos RTS/CTS y XON/XOFF.

El protocolo que vayamos a elegir dependerá del dispositivo con el que vayamos a conectar. Si especificamos *comRTSXOnXOff*, el resultado es que se admiten los dos protocolos.

Tabla 2.

Constante	Valor	Descripción
comEventBreak	1001	Se ha recibido una señal de interrupción.
ComEventCTSTO	1002	Ha expirado el tiempo de espera para emitir. La línea "Listo para emitir" (CTS) estaba en nivel bajo durante el número de milisegundos indicado en *CTSTimeout* cuando se intentaba transmitir un carácter.
ComEventDSRTO	1003	Ha expirado el tiempo de espera de terminal datos preparados. La línea "Terminal de datos preparado" (DSR) estaba en nivel bajo durante el número de milisegundos indicado en *DSRTimeout*, cuando se intentaba transmitir un carácter.
ComEventFrame	1004	Error de trama. El hardware ha detectado un error de trama.
ComEventOverrun	1006	Puerto desbordado. No se leyó un carácter desde el hardware antes de llegar otro carácter, y el primero se ha perdido.
ComEventCDTO	1007	Ha expirado el tiempo de espera de detección de portadora. La línea "Detector de portadora" (CD) estaba en nivel bajo durante el número de milisegundos indicado en *CDTimeout* cuando se intentaba transmitir un carácter. Esta línea se denomina también Detector de señal en línea de recepción (RLSD).
comEventRxOver	1008	Desbordamiento del buffer de recepción. No hay espacio suficiente en el buffer de recepción.
comEventRxParity	1009	Error de paridad. El hardware ha detectado un error de paridad.
ComEventTxFull	1010	*buffer* de transmisión lleno. Se ha intentado colocar un carácter más en la cola mientras el *buffer* de transmisión estaba lleno.
ComEventDCB	1011	Error inesperado al obtener el bloque de control de dispositivo (DCB) para el puerto.

Listado 1.

Comando	Título
A/	Re-ejecuta comando
ATA	Responder
ATBn	Configura CCITT o Bell
ATCn	Control de portadora
ATDn	Marcar
ATE	Eco
ATFn	Seleccionar modulación de línea
ATHn	Conmuta a control de colgar
ATIn	Identificación
ATLn	Volumen del altavoz
ATMn	Control del altavoz
ATNn	Modulación de conexión
ATOn	Regresa a línea de estado
ATP	Configura pulsos por defecto
ATQn	Visualiza los códigos de respuesta
ATT	Configura los tonos por defecto
ATVn	Hoja de códigos de respuesta
ATWn	Informe de progreso de negociación
ATXn	Códigos de respuesta extendidos
ATYn	Control de desconexión de espacio largo
ATZn	Reset
AT&Cn	Opción DCD
AT&Dn	Opción DTR
AT&F	Restaura la configuración de fábrica
AT&Gn	Configura el tono de guarda
AT&Jn	Selecciona el jack de teléfono
AT&Kn	Control de flujo del Módem/DTE
AT&Ln	Tipo de línea
AT&Mn	Modo de comunicación
AT&Pn	Relación de marcación de pulsos
AT&Rn	Opción RTS/CTS
AT&Sn	Opción DSR
AT&Tn	Test y diagnóstico
AT&V	Ver configuración actual y del usuario
AT&Wn	Almacena la configuración del usuario
AT&Xn	Selecciona la fuente del reloj
AT&Yn	Designa la configuración del usuario por defecto
AT&Zn=x	Memoria el nº de teléfono
AT\An	Tamaño del bloque MNP máximo
AT\Bn	Transmite corte
AT\Gn	Control de flujo de Módem a Módem
AT\Kn	Control de corte
AT\Ln	Control de transferencia de bloque MNP
AT\Nn	Modo de control de operación
AT%L	Informe de nivel de señal recibida
AT%Q	Informe de calidad de señal de línea

Nota. Si establecemos este valor a *comRTS* o *comRTSXOnXOff*, debemos asignar la propiedad *RTSEnabled* a *True*. De lo contrario, podremos conectarnos y enviar datos, pero no recibirlos.

El evento OnComm y la propiedad CommEvent

Según el alcance y la funcionalidad de la aplicación, puede que necesitemos controlar y responder a una serie de eventos y errores que pueden producirse durante la conexión con otro dispositivo o en la recepción o transmisión de los datos.

El evento *OnComm* y la propiedad *CommEvent* permiten interceptar y comprobar el valor de los eventos y errores de comunicación.

Cuando se produce un evento o un error de comunicación, se desencadena el evento *OnComm* y se modifica el valor de la propiedad *CommEvent*.

De esta forma, si es necesario, tendremos la posibilidad de comprobar el valor de *CommEvent* cada vez que ocurra el evento *OnComm*.

Como las comunicaciones (especialmente a través de líneas telefónicas) pueden ser impredecibles, la interceptación de estos eventos y errores permite responder a ellos adecuadamente.

En la tabla siguiente se enumeran los eventos de comunicaciones que desencadenarán el evento *OnComm*. Los valores indicados se escriben en la propiedad *CommEvent*:

El evento *OnComm* también se desencadena, y se escribe un valor en la propiedad *CommEvent*, cuando ocurren los errores expuestos a continuación.

El siguiente ejemplo muestra cómo controlar los eventos y errores en la comunicación. Después de cada instrucción **Case** relacionada, podemos insertar código para controlar un error o evento en particular.

```
Private Sub MSComm_OnComm ()

Select Case MSComm1.CommEvent
```

Si se producen errores durante la conexión, el evento OnComm y la propiedad ComEvent, permiten interceptar dichos fallos

Controlar cada evento o error escribiendo código en cada instrucción Case

```
Errores
  Case comBreak        ' Se ha recibido una interrupción.
  Case comEventCDTO    ' Tiempo de espera CD (RLSD).
  Case comEventCTSTO   ' Tiempo de espera CTS.
  Case comEventDSRTO   ' Tiempo de espera DSR.
  Case comEventFrame   ' Error de trama
  Case comEventOverrun ' Datos perdidos.
  Case comEventRxOver  ' Desbordamiento del buffer de recepción.

  Case comEventRxParity ' Error de paridad.
  Case comEventTxFull   ' buffer de transmisión lleno.
  Case comEventDCB      ' Error inesperado al recibir DCB

' Eventos
  Case comEvCD      ' Cambio en la línea CD.
  Case comEvCTS     ' Cambio en la línea CTS.
  Case comEvDSR     ' Cambio en la línea DSR.
  Case comEvRing    ' Cambio en el indicador de llamadas.
  Case comEvReceive ' Recibido nº RThreshold de caracteres.
  Case comEvSend    ' Hay un número SThreshold de
' caracteres en el buffer de transmisión.

  Case comEvEof     ' Se ha encontrado un carácter
' EOF en la entrada.

End select
End Sub
```

Comunicándose con el módem

En las siguientes líneas vamos a observar otro ejemplo que está creado especialmente con la finalidad de establecer una comunicación simple con el módem. El propósito del mismo es efectuar la apertura y cierre del puerto serie, además de enviar comandos en forma de caracteres al módem:

```
Private Sub Form_Load ()

    ' buffer para almacenar la cadena de entrada
    Dim Instring As String

    ' Utilizamos COM2 para efectuar las comunicaciones
    MSComm1.CommPort = 2

    ' 9600 baudios, sin paridad, 8 bits de datos y 1 bit de parada.
    MSComm1.Settings = "9600,N,8,1"

    ' Indicamos al control que lea todo el buffer al usar Input.
    MSComm1.InputLen = 0

    ' Abrimos el puerto.
    MSComm1.PortOpen = True

    ' Enviamos al módem el comando de atención "AT"
    MSComm1.Output = "AT" + Chr$(13)

    ' Esperamos a que vuelvan los datos al puerto serie.

Do
    DoEvents

Loop Until MSComm1.InBufferCount >= 2

' Leemos los datos de respuesta de aceptación en el puerto serie.
    Instring = MSComm1.Input

    ' Cerramos el puerto serie.
    MSComm1.PortOpen = False

End Sub
```

Nota: El control *MSComm* puede usar el método de sondeo o el controlado por eventos para recuperar datos del puerto. Este sencillo ejemplo usa el método de sondeo. En el ejemplo anterior, se utilizaba el método controlado por eventos para la detección de errores en la comunicación.

Comandos compatibles con Hayes

Para poder enviar comandos al módem debemos conocer la sintaxis de los mismos. Por norma general, la mayoría de los modems compatibles con el estándar Hayes utilizan una serie de comandos universales.

Dependiendo del tipo de módem utilizado, así como del fabricante, estos comandos pueden variar ligeramente de unos a otros. La mejor referencia de los comandos Hayes soportados por nuestro módem la podemos encontrar en el manual del fabricante.

De todos modos, en el cuadro se muestra una lista orientativa con los comandos AT para los modems/fax basados en uno de los fabricantes más prestigiosos: Rockwell.

suscríbete

a Sólo Programadores

y consigue un magnífico descuento

suscripción normal

ahorro **20%**

12 revistas (1 año) por sólo...

9.350 ptas.

suscripción estudiantes (carreras técnicas)

ahorro **40%**

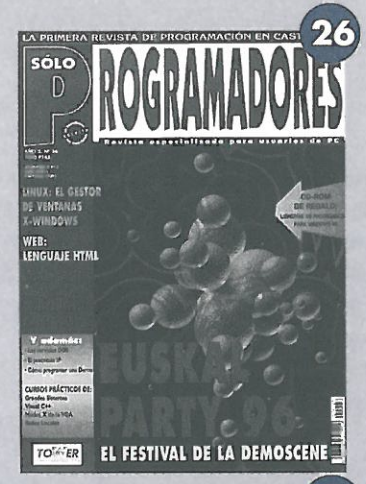
12 revistas (1 año) por sólo...

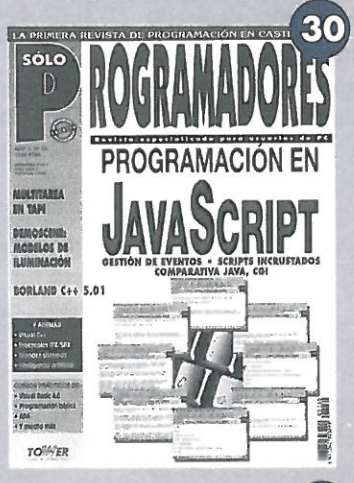
7.050 ptas.

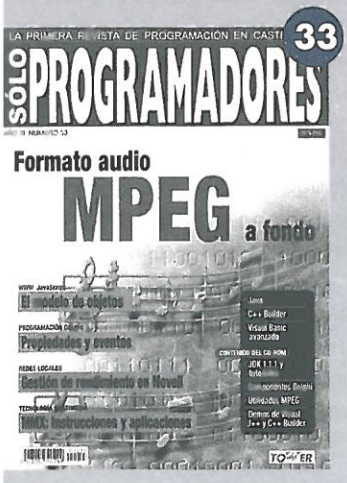
números atrasados

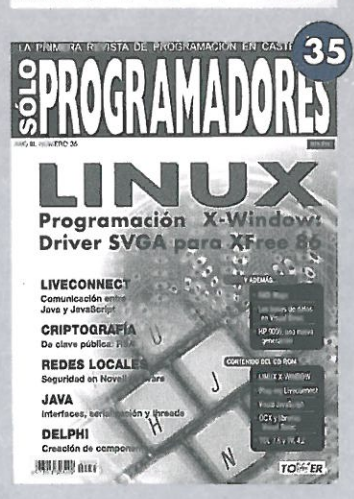
SÓLO PROGRAMADORES

completa ya tu colección









WINDOWS NT SERVER

Internet Information Server (I): Configuración básica

Enrique Díaz Trobo (enriqued@jet.es)

Este mes añadiremos la aplicación Internet Information Server a nuestra intranet basada en Windows NT. Se trata de un servidor tres en uno, mediante el cual podemos dar servicio WWW, FTP y Gopher de forma simultánea. Todos estos servicios podrán ser utilizados por nuestros usuarios mediante cualquier navegador de Internet.

En este artículo abordaremos la administración básica de todos los servicios que ofrece Internet Information Server dejándolo listo para que funcione, sin grandes complicaciones. Más adelante profundizaremos en aspectos relacionados con la seguridad y la estructuración de nuestro centro servidor de Internet/intranet.

Realizar la instalación de Internet Information Server (a partir de ahora IIS) no representa problemas ya que se hace de forma conjunta con la Windows NT. Incluso si decidimos en su momento no instalarlo, quedará en la barra de inicio un acceso directo hacia el programa de instalación. Si no disponemos de esta posibilidad se podrá instalar con el icono *Agregar o quitar programas* del *Panel de control*.

Una vez instalado, dispondremos de dos herramientas para su administración: el Administrador de servicios de Internet de Microsoft y otro de idéntico título, pero con el formato de una página HTML de nuestro servidor. Esto significa que podemos configurar y administrar el servicio desde cualquier punto de la red, sin necesidad de tener instalado un cliente RAS, ya que con el protocolo TCP/IP y un navegador de Internet tenemos suficiente.

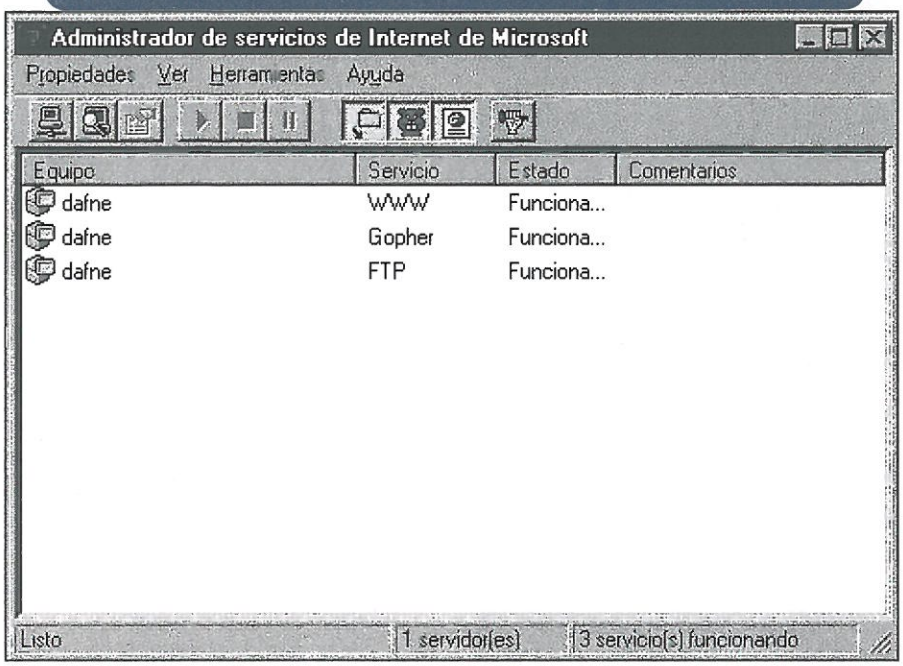
Por supuesto, también podemos administrar remotamente cualquier servidor IIS de la red, en caso de disponer de varios. Desde la ventana del Administrador del IIS podremos arrancar, detener o dejar en pausa los diferentes servicios. Si lo que deseamos es configurarlos para que arranquen (o no) durante el inicio del sistema, debemos hacerlo ejecutando el icono *Servicios* del *Panel de control*.

Configuración del servicio WWW

Para configurar cualquiera de los tres servicios haremos doble clic sobre ellos en la ventana del *Administrador de servicios de Internet de Microsoft*. En este caso lo haremos sobre el servicio WWW. Una vez efectuada dicha acción aparecerá una ventana con las diferentes opciones configurables, dentro de la solapa *Servicio*:

- Puerto TCP. Determina el puerto que está utilizando el servicio. Por defecto es el 80, aunque podemos cambiarlo por cualquier puerto TCP único. Si lo cambiamos deberemos reiniciar el equipo para que el cambio tenga efecto.
- Tiempo de espera de la conexión. Especifica el tiempo de inactividad en segundos que dejaremos transcurrir antes de desconectar al usuario.
- Máximo número de conexiones. Indica el número máximo de conexiones simultáneas que permitiremos en el servidor.
- Nombre de usuario y contraseña. Establecemos la cuenta de usuario

Figura 1.



que se utilizará para controlar los permisos de todas las conexiones anónimas. Por defecto es una cuenta creada en la instalación del IIS y cuyo patrón es IUSR_nombre_del_equipo; en nuestro caso IUSR_DAFNE.

Esta cuenta debe tener permiso de lectura sobre el directorio \WINNT\InetPub\wwwroot. Cuando se creó la cuenta de invitado, se le asignó una contraseña aleatoria que sólo se usa dentro de Windows NT y que no puede estar en blanco. Si cambiamos esta contraseña, deberemos cambiarla también en el apartado *Contraseña* para que coincidan tanto en el *Administrador de usuarios para dominios* como en el *Administrador de servicios de Internet*. Pese a ello, los usuarios anónimos no utilizarán un nombre de usuario y una contraseña.

La cuenta IUSR_DAFNE actúa a modo de *plantilla* para que podamos establecer lo que pueden y no pueden hacer los usuarios anónimos. Otro detalle importante es asegurarse de que esta cuenta tiene derechos para *iniciar sesión local* (se le concede este derecho de forma predeterminada). Si no vamos a permitir anónimos y lo que deseamos es permitir el acceso para algunos usuarios debemos concederles el derecho de *Iniciar sesión local*.

- Permitir anónimos. Éste indica principalmente si se van a permitir accesos anónimos al servidor.
- Básica. Especificamos que las contraseñas se enviarán sin codificar antes de la transmisión.
- Autentificación Desafío/Respuesta de Windows NT. Indica que se va a utilizar una codificación automática de los nombres de usuario y sus contraseñas.
- Comentarios. Especifica el texto descriptivo que se va a presentar en la pantalla principal del administrador de IIS.

Configuración de los directorios

En la solapa de *Directorios* estableceremos los siguientes valores: En *Directorio* se incluirá la ruta de los directorios utilizados, en *Alias* la ruta que se utiliza para los directorios virtuales, en *Dirección* se muestra la dirección IP para el servidor virtual que usa dicho directorio, mientras

Figura 2.

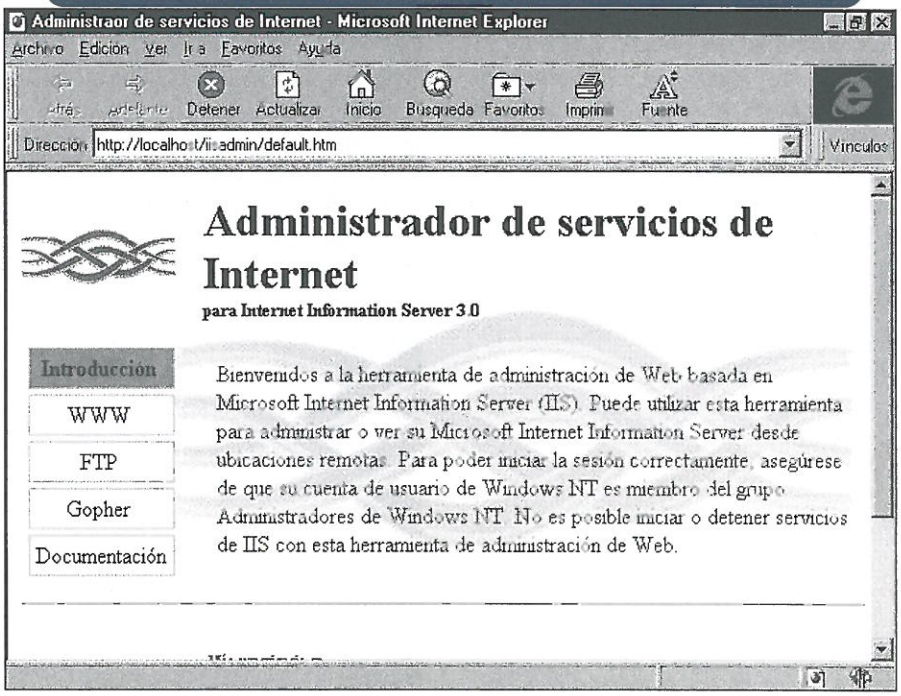
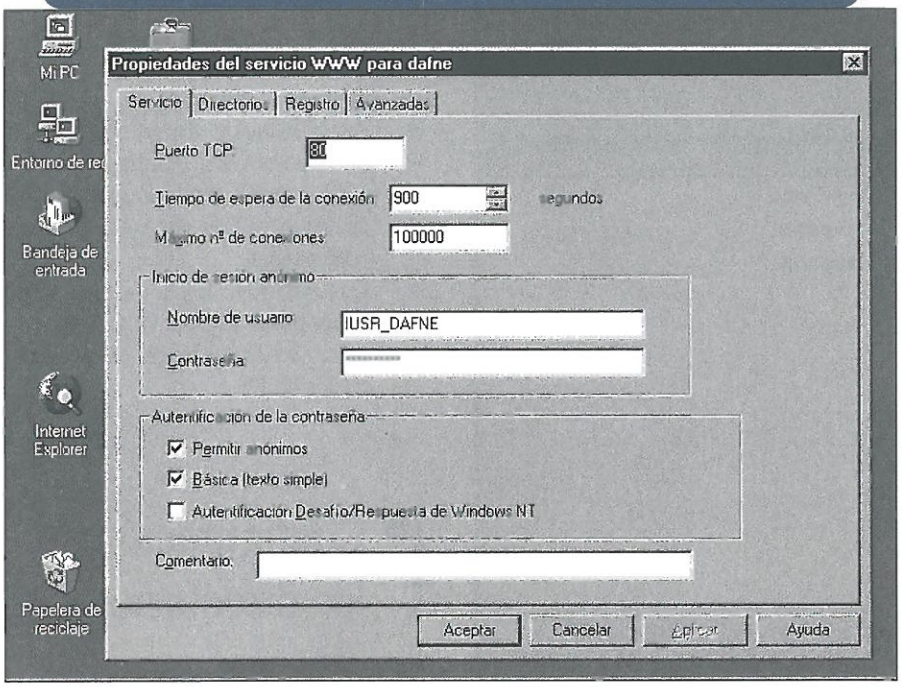


Figura 3.



que *Error* muestra los errores del sistema relacionados con dicho directorio.

Los apartados *Habilitar documento predeterminado* y *Se permite examinar directorios* se utilizan para establecer las presentaciones que aparecen si un usuario remoto no especifica un archivo concreto; es decir, cuando por ejemplo teclea http:\\dafne\ sin especificar un nombre de archivo. Por defecto está establecido el documento DEFAULT.HTM, de modo que éste será el documento que se presente a todos los usuarios que se conecten al servidor y no especifiquen ningún archivo. Por supuesto, podemos cambiarlo. Lo más usual es que se llame INDICE.HTML o INDEX.HTML, ya que es el criterio más extendido en el ámbito internacional, aunque igual de efectivo.

Si marcamos la solapa *Registro*, apreciaremos los siguientes apartados:

- Activar registro. Indica que se guardan en un archivo los datos de la actividad del servicio.

En el bloque *Registrar en archivo*, se indica que se almacenen en un archivo los datos de la actividad del servidor. Tenemos las siguientes opciones:

- Formato de registro, indica el tipo de formato de registro a utilizar.
- Abrir nuevo registro automáticamente. Indica la frecuencia con la que se generara un nuevo registro. Puede ser: diario, semanal, mensual, o cuando el archivo alcance un determinado tamaño.
- Directorio del archivo de registro. Muestra la ruta donde se encuentran los archivos de registro (el nombre de archivo está formado por un prefijo *in*, más los dos últimos dígitos del año, el número del mes, el número del día y la extensión LOG).

En el bloque registrar en base de datos SQL/ODBC, se indica que se guarden en una tabla ODBC los datos de la actividad. En ella se encuentran las opciones:

- Nombre de origen de datos ODBC(DSN). Indica el origen de datos ODBC que utilizar.
- Tabla. Indica el nombre de la tabla donde se guardarán los datos de la actividad realizada.
- Nombre de usuario. Indica un nombre de usuario válido para el equipo donde se encuentra la base de datos.
- Contraseña. Indica la contraseña de dicho usuario.

Figura 4.

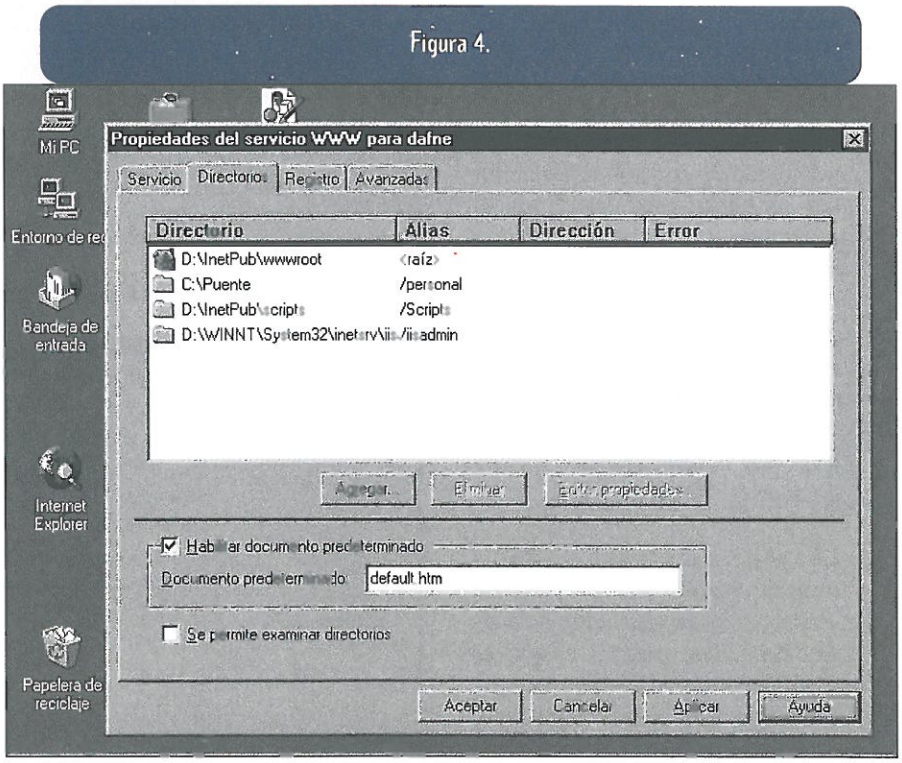
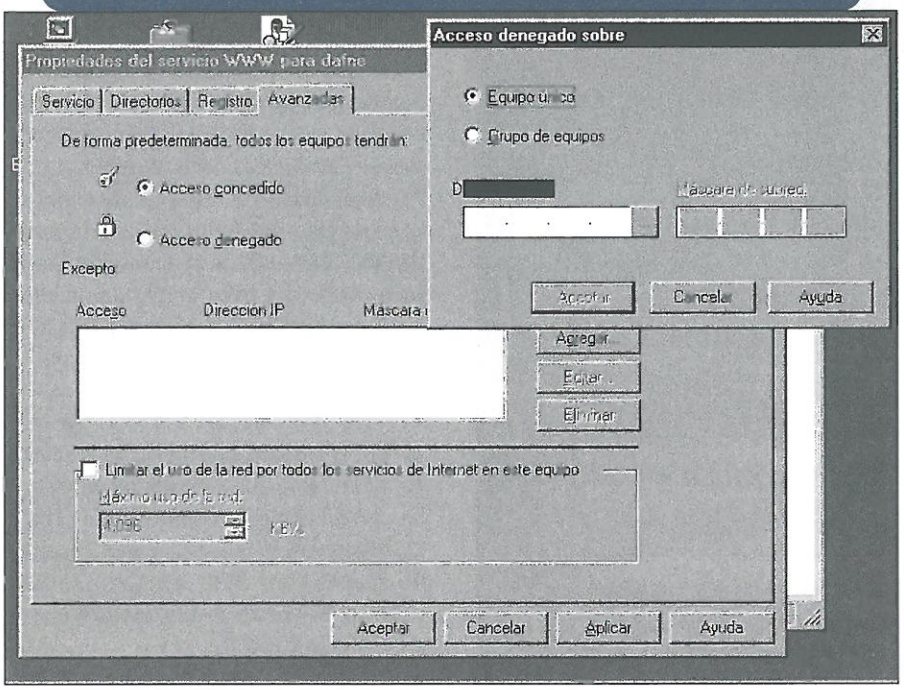


Figura 5



Si marcamos la solapa *Avanzadas* aparecerá una ventana similar parecido a la de la Figura 5.

En ella se muestran los apartados:

- Acceso concedido. Indica que todos los equipos podrán tener acceso a este servicio.
- Acceso denegado. Indica que todos los equipos tienen el acceso denegado a este servicio, excepto aquéllos que están indicados en la lista que rellenaremos pulsando Agregar e indicando la dirección IP pertenecientes a dichos equipos.
- Máximo uso de la red. Si marcamos limitar el uso de la red por todos los servicios de Internet en este equipo, debemos indicar el ancho de banda máximo que podrán utilizar los equipos para emplear este servicio.

Tras guardar la configuración, estamos listos para crear las páginas Web. Se visualizarán con cualquier explorador, indicando el URL http://nombre del equipo. Podremos especificar otros directorios, donde tengamos guardadas páginas que posean el formato HTMLl, por ejemplo http://dafne/personal/. No debemos olvidar que el nombre de la primera página debe coincidir con el especificado en *Habilitar documento predeterminado.*

Configuración del servicio de Gopher

Al igual que para configurar el servicio de WWW efectuaremos una doble pulsación sobre el servicio Gopher, en el *Administrador de servicios de Internet*. Veremos la pantalla de la Figura 6.

Bajo la pestaña *Servicio* encontramos los siguientes apartados:

- Puerto TCP. Determina el puerto que está utilizando este servicio. Por defecto es el 70, pero podemos cambiarlo por cualquier número de puerto TCP único.
- Tiempo de espera de la conexión. Indica el tiempo de inactividad en segundos que ha de pasar para que el servidor desconecte al usuario.
- Máximo número de conexiones. Indica el número máximo de conexiones simultáneas que se podrán establecer con el servidor.

En el bloque *Administrador del servicio* existen las siguientes opciones:

- Nombre. Establecemos la cuenta de usuario que se dispone a administrar el servicio Gopher.
- Dirección de correo electrónico. Indica la dirección de correo del administrador de Gopher.

En el bloque *Inicio de sesión anónimo* tenemos las siguientes opciones:

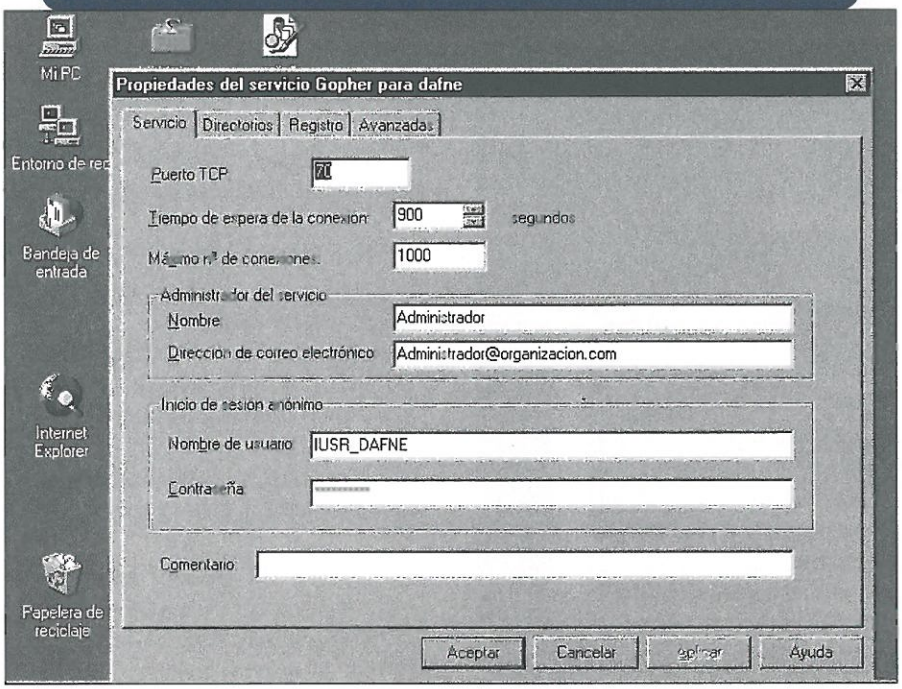
- Nombre de usuario. Establece la cuenta de usuario que se va a utilizar para controlar los permisos de todas las conexiones anónimas (la cuenta IUSR_DAFNE que se creó al instalar el IIS). Debemos comprobar que esta cuenta tenga permiso de lectura sobre c\winnt\inetpub\gophroot\
- Contraseña. Al crear la cuenta, Windows NT le asignó una contraseña aleatoria. Esta contraseña no puede estar en blanco y si la cambiamos debemos asegurarnos de que la contraseña coincida tanto en el *Administrador de usuarios para dominios* como en el *Administrador de servicios de Internet*.

Bajo la pestaña *Directorios*, podremos establecer los directorios que utilizará Gopher y un alias para los directorios virtuales. *Error* muestra los errores del sistema para dicho directorio.

La pestaña registro funciona igual que para el WWW.

Una vez configurado el servicio procederemos a la copia de los archivos que queramos incorporar al directorio c:\inetpub\gophroot. Por otro lado crearemos archivos de etiquetas con vínculos hacia archivos del mismo ordenador o de otros equipos de la red. Con ello tendremos la posibilidad de utilizar el servi-

Figura 6



cio de Gopher mediante cualquier navegador utilizando el URL gopher://dafne/.

- **Archivos de etiquetas de Gopher**

Los archivos de etiquetas se utilizan como suplemento a la presentación normal de Gopher y para proporcionar vínculos con otros equipos. Toda la información sobre un archivo enviada al cliente procede de los archivos de etiquetas, los cuales básicamente contienen la siguiente información:

Los servicios de Gopher y FTP también pueden utilizarse desde un navegador

- Nombre de presentación.
- Nombre del equipo.
- Número de puerto.

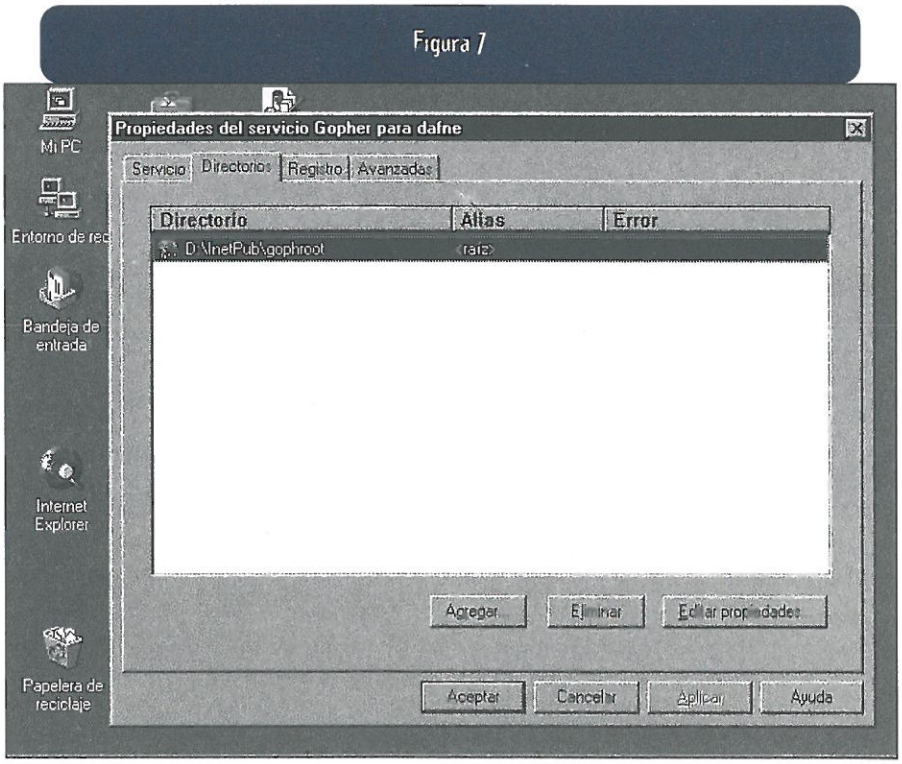
Disponemos de la posibilidad de agregar más cantidad de información a los archivos de etiquetas, tales como el nombre de usuario y el de correo electrónico del administrador del servidor, la fecha de creación del archivo y la fecha de la última modificación.

En aquellos discos que hayamos formateado con el sistema de archivos FAT, el nombre del archivo de etiquetas será el mismo que el archivo que describe con la extensión GTG añadido al nombre del archivo. Por ejemplo, si el archivo de contenido es INDICE.TXT, el nombre del archivo de etiquetas será INDICE.TXT.GTG. Podemos modificar estos archivos empleando un editor de texto ASCII. El comando **gdsset** crea éstos archivos con el atributo oculto activado, por lo que tendremos que quitárselo o bien utilizar la opción para ver este tipo de archivos.

En los discos formateados con NTFS, el nombre del archivo de etiquetas será el mismo que el archivo que describe, añadiéndole:gtg. Por ejemplo, INDICE.TXT tendrá su correspondiente nombre de etiquetas en INDICE.TXT:GTG. En este caso no es posible editar estos archivos con un editor ASCII.

Si hemos configurado el equipo utilizando el sistema de archivos FAT, tendremos que mover el archivo de etiquetas cuando movamos los archivos de datos correspondientes. Para crearlos utilizaremos el comando gdsset desde la línea de

Figura 7



comandos, siguiendo la sintaxis que se escribe a continuación:

```
gdsset <opción><archivo>
```

El comando aplicará automáticamente el atributo de oculto a los archivos de etiquetas generados.

Como puede observarse en la figura de la tabla 1 existe una lista básica de opciones, mientras que en la tabla 2 se muestra cómo aparecen los tipos de archivo que están disponibles.

Para crear una etiqueta llamada Principal de un archivo llamado leame.txt, escribimos la siguiente orden:

```
gdsset –g0 –f "Principal"
    c:\inetpub\gophroot\leame.txt
```

Este último código permite ver el contenido del archivo indicado.

```
gdsset –g0 –f "principal" –h manolo
    c:\inetpub\gophroot\archivo.txt.
```

Crea un vínculo con la etiqueta principal hacia un archivo de texto situado en un equipo llamado *manolo*.

Tabla 1.- Opciones para GDSSET

A "nombre"	El nombre del administrador. Si se omite su valor será el especificado en la configuración del servicio Gopher.
C	Se utiliza para modificar un archivo existente
E correo	Dirección de correo del administrador del servicio.
F "nombre"	Nombre amigable o descripción para el archivo.
G opción	Una de las opciones disponibles de la Tabla 2 para indicar el tipo de archivo. Si se omite se establecerá en 9.
H equipo	Nombre del equipo al que se va a vincular.
R	Muestra la información almacenada en el archivo de etiquetas.

Configuración del servicio FTP

La selección de este servicio se realiza de forma análoga a los descritos anteriormente. Dentro de la *Servicio* proporcionaremos un número de puerto TCP único. Para el resto, la configuración del servicio FTP funciona de forma similar a los servicios WWW y Gopher, aunque con alguna salvedad.

La opción *Permitir sólo conexiones anónimas* especifica que los usuarios no podrán iniciar sesión con su nombre de usuario y su contraseña. De esta forma, el servidor únicamente permitirá realizar conexiones anónimas (*ftp anonymous*).

También disponemos de la de *Mensajes*, en la que se establecerán los mensajes de bienvenida y salida del servicio de FTP, así como el mensaje que se mostrará cuando se alcance el número máximo de conexiones.

No es necesario activar los tres tipos de servicio, aunque se encuentren instalados, ahorrando así recursos del sistema

Enmarcados en la pestaña de *Directorios* estableceremos los directorios que utilizará el servicio de FTP.

En el apartado *Estilo* de la lista de directorios, indicaremos el formato en que queremos mostrar los archivos. Lo más normal y recomendable es mostrarlos en formato UNIX.

Los aspectos básicos de la configuración de los servicios se encuentran establecidos. Sólo resta indicar que no es necesario activar los tres servicios sólo porque se encuentren instalados y disponibles. Si realmente no hace falta utilizar alguno de ellos, resulta apropiado desactivarlo, ahorrando de esta manera recursos del sistema y tiempo de procesador.

Tabla 2.- Tipos de archivos

0	Un archivo, normalmente un archivo de texto sin formato.
1	Un directorio de Gopher.
2	Un servidor de directorio telefónico CSO.
3	Un error.
4	Un archivo BinHex de Macintosh.
5	Un archivo binario de MS-DOS.
6	Un archivo UNIX Uuencoded.
7	Un servidor de búsqueda indexada.
8	Una sesión Telnet.
9	Un archivo binario.
c	Un calendario o calendario de eventos.
g	Un gráfico de archivo de intercambio gráfico (GIF).
g	Una página hipertexto HTML de World Wide Web.
i	Un texto en pantalla que no es un elemento.
I	Otra clase de archivo de imagen.
m	Un archivo de buzón con formato BSD.
P	Un documento PDF.
T	Una sesión TN3270 con un sistema central.
:	Una imagen de mapa de bits

PROGRAMACIÓN CIENTÍFICA

Integración Numérica

José Antonio Mendoza Lorente (jmendoza@dit.upm.es)

Cuando una integral definida no tiene primitiva, o su cálculo resulta muy complicado por los métodos tradicionales, se recurre a la integración numérica. Ésta consiste en calcular un valor, lo más aproximado posible, al que obtendríamos mediante el cálculo integral.

Como sabemos, la integral de una función de una sola variable, a lo largo de un intervalo, da el valor del área comprendida por el eje de abscisas, la curva de la función, y dos rectas verticales que pasan por los extremos del intervalo, siempre y cuando la función subintegral no corte al eje de abscisas y sea mayor que cero, a lo largo del intervalo.

En caso contrario, el resultado será la suma de áreas, unas "negativas", las de debajo del eje y otras "positivas", las que están por encima de éste. Si el resultado es 0, no es que no haya área, es que unas con otras se cancelan. La idea de área es muy importante en integración numérica. Esto se debe al hecho, de que para hallar el valor aproximado, se busca un área equivalente a la de nuestra integral mediante rectángulos, trapecios u otras figuras geométricas.

Antes de empezar con el desarrollo del artículo, veamos la notación empleada en los cálculos matemáticos.

Debido a problemas en la reproducción de caracteres habitualmente utilizados en las ecuaciones matemáticas, en vez de emplear los símbolos tradicionales de integrales, sumatorios, y productos, es decir el "gusano", la sigma mayúscula y la pi mayúscula emplearemos los siguientes :

Para la integral, INT[a,b]{f(x)}, donde a y b son los extremos del intervalo y f(x) la función subintegral

El sumatorio lo indicaremos con SUM[i=a,b](g(i)), el índice del sumatorio varía entre a y b y g(i) es una expresión que depende de este índice.

Por último el producto será PRO[i=a,b](g(i)), lo mismo que en el sumatorio, pero esta vez en vez de sumar los términos los multiplicaremos.

Para comenzar con la integración numérica, veamos la expresión que resume su filosofía.

$$INT[a,b]\{f(x)\}=SUM[i=1,n]\{\alpha_i f(x_i)\} + R(f)$$

En esta expresión se observa la metodología que se va a seguir en la integración numérica, calcular la integral a partir de un sumatorio, y un resto, que llamaremos error.

Observamos a su vez que lo que sumamos es la función evaluada en unos determinados puntos, multiplicada por unos coeficientes. La ciencia pues está ahí para encontrar cuales son los puntos adecuados donde evaluar la función, y calcular los coeficientes que la multiplican de tal manera que obtengamos un recubrimiento del área lo más equivalente posible.

No todos los puntos son igual de buenos, su elección dependerá mucho de la naturaleza de la función que tengamos que integrar, pues no es lo mismo integrar una recta, que una función con muchos picos y valles en nuestro intervalo.

Los a_i y los x_i son la expresión última del proceso matemático seguido.

Este consiste en la sustitución de nuestra función por una función de interpolación que se caracteriza por valer lo mismo que nuestra función en una serie de puntos en los cuales vamos a evaluarla. En resumen, sustituiremos nuestra función por p(x) de tal manera que f(x) = p(x) + e(x), donde e(x) es el error cometido en la aproximación.

La integración numérica más empleada es la polinómica, aquella en la que la función p(x), que aproxima a la función subintegral es un polinomio. La función pues se descompone en un polinomio y un determinado error.

El presente artículo no trata de la interpolación de funciones, aunque para la integración numérica resulte necesario.

Sólo decir que se usará la interpolación de funciones mediante polinomios, o lo que es lo mismo intentaremos encontrar polinomios que sean lo más parecidos a nuestra función. Se usan polinomios, debido a que son funciones cuyo manejo resulta muy fácil. Existen muchas clases de polinomios de interpolación, los mas usados son: Taylor, Hermite, Lagrange, etc.

Todo lo anterior parece muy complicado pero simplificándolo al máximo, nos lleva a aproximar la integral por un producto el de (b-a) por $f(x_0)$. Este x_0 puede ser a, b o cualquier otro punto del intervalo, el que mejor nos parezca.

Habrá unos que comentan más error y otros que cometan un error inferior. Enlazando este ejemplo con la teoría diremos que nuestro polinomio de interpolación será una constante $f(x_0)$, la constante es un polinomio de grado cero, y que $a_i = 1$.

Podríamos pensar que de esta manera se comete gran cantidad de error, lo cual es realmente cierto. Sin embargo, si por ejemplo, nuestra función fuera una recta, el error cometido resultante de esta operación sería cero. Una recta horizontal la aproximaríamos por su valor que es el mismo para todos los puntos del intervalo, error cero.

Una recta inclinada, la aproximaríamos por el valor que toma en el punto medio del intervalo, y también cometeríamos en este caso error cero.

El error que cometemos a la izquierda del punto medio, lo compensamos con el que podemos cometer a la derecha, ya que si uno es por defecto el otro será por exceso.

Con este sencillo ejemplo hemos visto la importancia que tiene la elección del punto donde evaluamos la función.

El otro aspecto es el tipo de polinomio que usamos para sustituir a nuestra función. En el caso anterior nos sirvió uno de grado cero ya que nuestra función era una simple recta. Cuando esta se complica nos harán falta polinomios más complejos, más adaptados a nuestra función.

Otro aspecto de la integración numérica es la división que podemos hacer del intervalo de integración. Cuanto mayor es el intervalo, mayor puede ser el error cometido.

Una forma de reducirlo es dividir el intervalo en partes mas pequeñas, en las cuales la evaluación de la integral resulte mucho más exacta.

La integral entre a y b es la suma de otras dos, una entre a y c y otra entre c y b siempre y cuando a<c<b.

Por lo tanto atacaremos el problema desde distintos frentes.

Uno dividiendo el intervalo en un conjunto de subintervalos, y otro tomando varios puntos dentro de cada subintervalo, lógicamente los adecuados, y por último construyendo un buen polinomio interpolador, que sea lo mas parecido a nuestra función.

Queda hablar, con respecto a esta cuestión concreta, del error cometido. Este suele ser un punto delicado, pues de su cantidad y naturaleza depende que tengamos una buena aproximación, y lo que es más importante, tener una expresión que lo cuantifique.

Si para conocer el error cometido restamos al valor real al aproximado, hemos perdido un gran esfuerzo en calcular un valor aproximado del que ya conocemos su valor real.

El cálculo de la expresión del error necesita de complejos desarrollos matemáticos que escapan a la intención de este artículo. Para profundizar se indican temas adjuntos al final la bibliografía.

Para cada función, intervalo y polinomio interpolador aparece una expresión distinta. En este articulo veremos dos métodos de integración numérica el de Simpson y el de Gauss-Legendre.

Para cada uno de ellos daremos la expresión de su error, pero sin profundizar demasiado en este aspecto de la integración. Y como mejora de ambos métodos usaremos también la cuadratura adaptativa, que consiste básicamente en dividir nuestro intervalo tantas veces como sea necesario, como para que en cada uno de ellos el error cometido no supere una cota preestablecida.

Cuadratura adaptativa basada en Simpsom

Antes de hablar del método de Simpson, debemos indicar que forma parte de una familia de fórmulas, las de Newton-Cotes.

Éstas a su vez pueden ser de dos clases, las abiertas y las cerradas. Las primeras no incluyen a los extremos del intervalo, y las segundas sí.

La de Simpson es aquélla en la que se toman tres puntos del intervalo, los extremos y el punto medio.

Por lo tanto en vez de integrar nuestra función, integraremos el polinomio interpolador de Lagrange :

$$INT[a,b]\{f(x)\} \approx INT[a,b]\{p(x)\}$$

donde :

$$p(x) = SUM[i=0,n]\{f(x_i)\ l(x_i)\}$$

y los l_i son :

$$l_i(x) = PRO[j=0,n]\{(x-x_j)/(x_i-x_j)\} \text{ ; con } j \neq i$$

Sustituyendo la expresión de los l_i en la de p(x) llegamos a la expresión de los α_i.

$$INT[a,b]\{f(x)\} \approx INT[a,b]\{p(x)\} = INT[a,b]\{SUM[i=0,n]\{f(x_i)\ l_i(x)\}\}$$

Metiendo la integral dentro del sumatorio :

$$INT[a,b]\{f(x)\} \approx INT[a,b]\{p(x)\} = SUM[i=0,n]\{\ f(x_i)\ INT[a,b]\{l_i(x)\ \}\} = =SUM[i=0,n]\{\ \alpha_i\ f(x_i)\ \}$$

A primera vista los l_i parecen difíciles de entender, pero si nos fijamos bien sólo valen uno para $x=x_i$ y cero para los demás x_j con lo que se consigue que función y polinomio interpolador tengan el mismo valor en dichos puntos.

Lógicamente en otros valores de x distintos de los puntos de muestreo la función y el polinomio no coincidirán. Pero a cambio de tener una f(x) sin primitiva ahora lo que tenemos es un cómodo polinomio por integrar.

La fórmula de Simpson nos dice que la integral de una función en el intervalo [a,b], es aproximadamente igual a :

$$INT[a,b]\{f(x)\} \approx [(b-a)/6] * [f(a) + 4f((a+b)/2) + f(b)]$$

Vemos en la expresión los puntos donde se evalúa la función subintegral, los extremos y el punto medio del intervalo de integración, y los coeficientes que multiplican a dichos valores. Los puntos, cumplen con la condición de estar x-espaciados. Los a_i vienen de la integración del polinomio de interpolación de Lagrange para cada punto de evaluación. Para profundizar más en el tema interpolatorio al final del artículo existe bibliografía relacionada con este aspecto, como ya se ha mencionado anteriormente.

Lo importante de los métodos numéricos de cálculo es conocer la naturaleza del error que cometen

La cuadratura adaptativa, la añadimos al proceso de cálculo, con el objeto de cometer menor error, la cual consiste en dividir el intervalo de integración en otros mas pequeños:

$$INT[a,b]\{f(x)\} = SUM[i=1,n]\{\ INT[x_{i-1},x_i]\{f(x)\}\}\ ;\ x_0 = a\ , x_n = b$$

Combinando ambas llegamos a:

$$INT[a,b]\{f(x)\} \approx SUM[i=1,n]\{[(x_i-x_{i-1})/6] * [f(x_{i-1}) + 4f((x_i+x_{i-1})/2) + f(x_i)]\ \}$$

Por lo que ya tenemos descompuesta nuestra integral en un sumatorio de integrales, cada una de ellas sobre una porción del intervalo original.

Planteados los fundamentos nos queda ver como automatizar el proceso de cálculo, y como acotar el error.

No olvidemos, que este, en nuestro caso queda en función de la derivada cuarta de la función subintegral (en un texto breve como el que nos ocupa resulta imposible mostrar todo el desarrollo matemático necesario, para exponer con integridad todo el tema, así como para dedicarse a los aspectos más complejos).

La fórmula de Simpson incluido el error cometido tiene la expresión que se indica en las siguientes líneas.

$$INT[a,b]\{f(x)\} = [(b-a)/6] * [f(a) + 4f((a+b)/2) + f(b)] - 1/90\ (b-a)^5\ f^{(4)}\ (\xi)$$

Para abreviar llamaremos S(a,b) a :

$$S(a,b) = [(b-a)/6] * [f(a) + 4f((a+b)/2) + f(b)]$$

Por lo que tendremos :

$$INT[a,b]\{f(x)\} = S(a,b) - 1/90\ (b-a)^5\ f^{(4)}\ (\xi) \quad (0)$$

Y llevándolo a la fórmula de la cuadratura adaptativa:

$$INT[a,b]\{f(x)\} = SUM\ [i=1,n]\{S_i+R(f)_i\}$$

Donde :

$$R(f) = - 1/90\ (b-a)^5\ f^{(4)}\ (\xi)$$

Vemos que para poder conocer el error es necesario calcular la derivada cuarta de la función, y un punto x del intervalo que sirva como punto cota de todos los del intervalo, que no es precisamente agradable.

Para evitarlo emplearemos un truco. Intentaremos poner el error en función de S(a,c), S(c,b), y S(a,b), donde c es el punto medio del intervalo: c =(a+b)/2.

Dependiendo de lo buena que sea la aproximación de una función por otra el tiempo de cálculo se reduce drásticamente

Si el error que se produce al integrar una función en un subintervalo [a,b] es superior a nuestras necesidades, lo que haremos será dividir el intervalo en dos, siendo este el fundamento de la cuadratura adaptativa.

Si a los resultantes les ocurriese lo mismo seguiríamos subdividiendo los intervalos hasta alcanzar el error requerido. Aunque la cuadratura adaptativa no exige que los subintervalos se dividan por

la mitad, lo haremos así, aunque se puede tomar cualquier otra, con tal de que se recubra el intervalo sin solapas. Para poner el error en función de las tres eses procederíamos así:

$$I = INT[a,b]\{f(x)\} = S(a,b) - 1/90\,(b-a)^5\,f^{4)}(\xi) \quad (1)$$

$$c = (a+b)/2$$

$$INT[a,b]\{f(x)\} = INT[a,c]\{f(x)\} + INT[c,b]\{f(x)\}$$

$$INT[a,c]\{f(x)\} = S(a,c) - 1/90\,(c-a)^5\,f^{4)}(\xi_1)$$

$$INT[c,b]\{f(x)\} = S(c,b) - 1/90\,(b-c)^5\,f^{4)}(\xi_2)$$

$$INT[a,b]\{f(x)\} = S(a,c) + S(c,b) - 1/90\,(c-a)^5\,f^{4)}(\xi_1) - 1/90\,(b-c)^5\,f^{4)}(\xi_2)$$

Sustituyendo c por su valor y operando:

$$I = INT[a,b]\{f(x)\} = S(a,c) + S(c,b) - 1/90\,((b-a)/2)^5\,(f^{4)}(\xi_1) + f^{4)}(\xi_2)) \quad (2)$$

Observamos que ambas expresiones se parecen bastante la (1) y la (2), solo las derivadas cuartas de la función subintegral evaluadas en puntos diferentes, y el un medio elevado a la quinta.

Para lo cual se hace la suposición de que la derivada cuarta permanece constante en los intervalos de integración, por lo que podemos decir que:

$$2*f^{4)}(\xi) = f^{4)}(\xi_1) + f^{4)}(\xi_2) \quad (3)$$

de lo que se deduce que:

$$I-S(a,b) = -1/90\,(b-a)^5\,(f^{4)}(\xi) \quad (4)$$

$$I-S(a,c) - S(c,b) = -1/(90\cdot 2^5)(b-a)^5\,[f^{4)}(\xi_1) + f^{4)}(\xi_2)] \quad (5)$$

eliminando los términos de error entre las ecuaciones 3,4, y 5 llegamos a:

$$I = INT[a,b]\{f(x)\} \approx S(a,c) + S(c,b) - 1/15\,[S(a,c)+S(c,b)-S(a,b)] \quad (6)$$

Ya no tenemos pues el factor de la derivada cuarta, que nos hubiera dificultado mucho los cálculos, y lo que es más importante, este método es recursivo, podemos dividir y dividir los subintervalos, hasta obtener el resultado adecuado, o lo que es lo mismo a un error prefijado.

Únicamente queda mostrar cual es la condición de división del intervalo de integración, y de los generados en otros a su vez más pequeños. Partiremoscon ello de un error máximo que le exigiremos al resultado. Sí lo conseguimos sin dividir el intervalo se resuelve el problema en caso contrario haremos más cantidad de divisiones.

Veamos la condición de división y su cálculo:

$$INT[a,b]\{f(x)\} = SUM[i=1,n]\{S_i+R(f)_i\} = SUM[i=1,n]\{S_i+ei\} = SUM[i=1,n]\{S_i\} + SUM[i=1,n]\{e_i\}$$

el error total será la diferencia entre valor real y valor aproximado:

$$e = INT[a,b]\{f(x)\} - SUM[i=1,n]\{S_i\}$$

y e_i el error de cada subintervalo proporcional a su anchura :

$$ABS(ei) <= e((x_i-xi-1)/(b-a)) \quad (7), \quad ABS(\text{valor absoluto})$$

y de (6) claramente el e_i es:

$$ABS(e_i) = 1/15\,(S(a,c)+S(c,b)-S(a,b)) \quad (8)$$

Este error es el error ponderado. A mayor anchura de subintervalo mayor error le corresponde. De (7) y (8) llegamos a la siguiente fórmula:

$$1/15[S(a,c)+S(c,b)-S(a,b)] < [e(x_i-x_{i-1})]/(b-a)$$

con el signo de mayor estricto se consigue mayor precisión.

Finalizada esta cuestión resta explicar el funcionamiento del programa adjunto. Se introducen los datos, coeficiente de la exponencial, grado del polinomio y sus coeficientes, los extremos a y b del intervalo de integración y el error total exigido al resultado, por último el número máximo de subdivisiones consecutivas. El proceso que sigue el programa es el siguiente. Evalúa la integral en [a,b] si la condición de error se cumple ya tenemos la solución si no divide el intervalo en dos [a,c] y [c,b], donde c es el punto medio del intervalo, y ahora el nuevo intervalo a evaluar es el [c,b]. Si no se satisface la condición de error vuelve a dividir [c,e] y [e,b] y vuelve a evaluarse este último.

Cuando la condición de error se satisface con la suma al total, la aportación de dicho intervalo evalúa el intervalo más próximo por la izquierda, repitiéndose el proceso hasta que no queden intervalos para evaluar. Lógicamente pretendemos que nuestro programa realizase la integral de cualquier función subintegral.

Esto se puede conseguir de dos maneras. La primera y tosca cambiando las líneas de código donde aparece la función subintegral. Y la otra, la elegante y efectiva, mediante un scanner y un parser, que analicen léxica y sintácticamente la expresión introducida para pasar de una serie de caracteres a una expresión evaluable por el ordenador. Lex y Yacc son dos herramientas del mundo Unix que pueden ser muy útiles para este fin. Aunque siempre hay seguidores que prefieren programar dichas rutinas a cara descubierta.

Cuadratura Adaptativa basada en Gauss-Legendre

Todo el desarrollo visto para Simpson vale para Gauss-Legendre, sólo difiere en que ahora la integral se aproxima por la fórmula de Gauss-Legendre. Es decir, ahora se emplean los polinomios de Legendre para encontrar los puntos "mágicos" donde evaluar la función. Si antes eran x-espaciados ahora serán las raíces de dichos polinomios. En los métodos de Gauss las funciones subintegrales vienen ponderadas por otras. En Legendre la función de peso empleada es

la unidad. La función de peso es la que multiplica a la función subintegral. En Simpson tomamos tres puntos, los extremos y el punto medio, ahora también tomaremos tres, las raíces del polinomio de Legendre de tercer grado, ±(3/5)^.5 y 0.

Los a_i se calculan como antes pero en vez de tener $l_{i(x)}$ ahora estarán estos al cuadrado. Además tendremos otra gran diferencia, la del intervalo, que si antes era cualquier a y b con b > a, en la fórmula de Gauss-Legendre es [-1,1]. Esto no representa ningún problema, pues con un simple cambio de variable llevamos cualquier intervalo [a,b] a [-1,1]. Dicho cambio es:

$$x = ((b-a)/2)\,t + (b+a)/2 \quad (9)$$

Resumiendo, la fórmula de Gauss-Legendre es:

$$I = INT[-1,1]\{f(x)\} = 5/9\, f(-(3/5)^{1/2}) + 8/9\, f(0) + 5/9\, f((3/5)^{1/2}) + 1/15750\, f^{(6)}(\xi)$$

Si aplicamos el cambio de variable (9) a:

$$I = INT\,[a,b]\{f(x)\}$$

Llegamos a:

$$I = INT\,[a,b]\{f(x)\} = ((b-a)/2) * INT[-1,1]\{f(t)\} =$$
$$= ((b-a)/2)[\,5/9\, f\{(b-a)/2(-(3/5)^{1/2})+(a+b)/2\} + 8/9 f\{(a+b)/2\} + 5/9\, f\{(b-a)/2((3/5)^{1/2}) + (a+b)/2\} +$$
$$+ 1/15750\, g^{(6)}(\xi)\,]$$

o lo que es lo mismo:

$$I = INT[a,b]\{f(x)\} = (b-a)/2\; INT[-1,1]\{f(t)\} = (b-a)/2\,[\,S(a,b) + 1/15750\, g^{(6)}(\xi)]$$

Observamos la gran semejanza de esta ecuación con (0).

Para llegar al mismo resultado que con Simpson, nos queda comentar la derivada sexta de g, por lo demás todo idéntico: la misma suposición de error constante, la misma ecuación de decisión de error, lógicamente con otros coeficientes, pero la estructura igual. Para eliminar nuestra dependencia del error, haremos igual que anteriormente, pero antes, veremos lo de la derivada sexta, $g^{(6)}(t)$, tiene dentro lo del cambio de variable. Al derivar seis veces nos sale (b-a)/2 a la sexta.

$$g^{(6)}(t) = ((b-a)/2)^{6}\,(\,((b-a)/2)\,t + (b+a)/2\,)$$

Veamos más de cerca el desarrollo:

$$I = INT[a,b]\{f(x)\} = I_1 + I_2 = INT[a,c]\{f(x)\} + INT[c,b]\{f(x)\}$$

Ahora es necesario realizar el cambio de variable, para llevar a todas las integrales al intervalo [-1,1], donde tiene lugar la integral de Gauss-Legendre. Recordemos que c es punto medio del intervalo c=(a+b)/2. Aplicando por tanto el cambio de variable llegamos a:

$$I = INT[a,b]\{f(x)\} = (b-a)/2\; INT[-1,1]\{f(t)\} = S(a,b) + ((b-a)/2)(1/15750)\, g^{(6)}(\xi)]$$

$$I_1 = INT[a,c]\{f(x)\} = (c-a)/2\; INT[-1,1]\{f(t)\} = S(a,c) + ((c-a)/2)(1/15750)\, g^{(6)}(\xi_1)]$$

$$I_2 = INT[c,b]\{f(x)\} = (b-c)/2\; INT[-1,1]\{f(t)\} = S(c,b) + ((b-c/2))(1/15750)\, g^{(6)}(\xi_2)]$$

$$S(a,b) = ((b-a)/2)[\,5/9\, f\{(b-a)/2(-(3/5)^{1/2}) + (a+b)/2\} + 8/9\, f\{(a+b)/2\} + 5/9\, f\{(b-a)/2((3/5)^{1/2}) + (a+b)/2\}\,]$$

tendríamos la misma expresión para [a,c] y [c,b], pero cambiarían la a y la b por la c según proceda.

En cuanto a los errores en función de las integrales, partiendo de la suposición de que la derivada sexta permanece constante, derivando seis veces, por lo que aparece la mitad del intervalo a la sexta:

$$I = INT[a,b]\{f(x)\} \approx S(a,c) + S(c,b) + 1/63[S(a,c) + S(c,b) - S(a,b)]$$

Y la condición de error que nos servirá, como en el caso anterior, para dividir o no un intervalo dependiendo de si supera la cota de error o no.

$$1/63[S(a,c)+S(c,b)-S(a,b)] < (e(x_i - x_{i-1}))/(b-a)$$

No queda nada más que comentar. Respecto al funcionamiento del programa, es idéntico al de Simpson, requiriendo el mismo tipo de datos referidos antes.

Comparativa de los dos métodos

Gauss-Legendre es manifiestamente superior a Simpson, tanto en el número de subdivisiones que realiza como en el tiempo de ejecución.

He llegado a esta conclusión integrando la función, exp(-(x+1)) en el intervalo [-11,11], con una precisión de 1E-13. Mientras Simpson se paraliza, Legendre realiza los cálculos en 25 segundos.

También es necesario mencionar que Gauss con sólo 37 subdivisiones lo consigue. Simpson para calcularlo necesita entre mil y diez mil. Mil no le son suficientes y con diez mil parece funcionar.

Estos valores se han obtenido con un ordenador con las siguientes características: Pentium 200Mhz MMX con 64 Megas de memoria RAM y Windows 95 como sistema operativo.

Para terminar, indicar que en el CD-ROM se encuentran los fuentes de los dos métodos, y para calcular cualquier otra integral que no tenga el aspecto de la exponencial de un polinomio, solamente basta cambiar las líneas de código que sustituyen una expresión por otra.

Bibliografía

- "CÁLCULO NUMÉRICO I" Autor: Mariano Gasca González editorial UNED.
- "ANÁLISIS NUMÉRICO, LAS MATEMATICAS DEL CALCULO CIENTIFICO" Autores : David Kincaid y Ward Cheney, editorial : Addisson-Wesley Iberoamericana.

BUSCAMOS PROFESIONALES

Si te gusta escribir y tienes experiencia en algunos de estos campos de la informática

HARDWARE:

Tarjetas gráficas, modems, PC Cards, cámaras digitales, escáneres, tarjetas de sonido, impresoras, monitores, pantallas planas, dispositivos de almacenamiento, vídeo digital...

SOFTWARE:

Ofimática en general, aplicaciones profesionales de diseño, Internet, autoedición, multimedia, animaciones 3D, programación, sonido, edición de vídeo, antivirus, edición Web, juegos...

INTERNET:

Navegación, videoconferencia, conectividad, redes, ActiveX, *plug-ins*, Shockwave, RDSI, sistemas abiertos, comercio electrónico, sistemas de seguridad, intranet, extranet...

Envíanos tu C.V. a la siguiente dirección

Tower Communications
Apartado de correos 54.283
28080 - Madrid

(Ref.: Hard - Soft - Int)

Correo del lector

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Pregunta

Hola, ante todo discúlpenme por el mail, pero creo que es un buen lugar para consultarles una duda que tengo.

Principalmente les escribo porque estoy bastante interesado en averiguar si existe un lenguaje de programación para los PC´s llamado PowerBuilder. Toda la información se refiere fundamentalmente a los Macintosh. Si es así, me gustaría saber que versión existe actualmente, y donde podría obtener información acerca de cómo adquirirlo. Por otro lado me gustaría saber si existen urls de foros que traten este mismo tema.

Muchas Gracias por anticipado.

Un saludo.

Respuesta

La verdadera plataforma de origen del PowerBuilder es el PC. Gracias a ella se pueden diseñar interfaces gráficas para el manejo de RDBMS (sistemas de bases de datos relacionales) sin tener que usar para ello lenguajes 3GL (de tercera generación) como podrían ser el C o el C++.

El efecto deseado de PowerBuilder es el de poder crear al mismo tiempo que se ve, el prototipo de la herramienta que más tarde crearíamos en un lenguaje 3GL. Este modo de trabajar es la llamada RAD (diseño rápido de aplicaciones). Disminuye los costes de diseño e implementación en la industria del software.

La actividad en este entorno 4GL es más sencilla que antes. Nos valdremos de ventanas, paletas y herramientas visuales que luego enlazaremos con un sencillo lenguaje de programación de guiones (scripts) llamado PowerScript. La propietaria de PowerBuilder es Sybase, cuyo Web dispone de un apartado de *downloads* específico para cada entorno, especialmente PC/Unix/NT. La dirección en la red es: http://www.sybase.com. La última versión del entorno es PowerBuilder 6.0.

La más extendida es la versión 5.0 para la que existen toda clase de utilidades y tutoriales. Respecto a los forums o grupos de news los más importantes son:

comp.soft-sys.powerbuilder dedicado exclusivamente a este tema.

En el próximo número de la revista se dedicará un apartado sobre la programación en PowerBuilder.

Figura 1.

CONTENIDO DEL CD

Las últimas novedades de software de desarrollo

Modula-3

Modula-3 es un miembro de la familia compuesta por lenguajes del Pascal. Este código diseñado a finales de los ochenta, Modula-3 corrige la mayoría de las deficiencias del Pascal y Modula-2 en la Ingeniería del software a un nivel práctico.

En particular, Modula-3 mantiene la simplicidad de los primeros lenguajes al mismo tiempo que proporciona facilidades para el manejo de excepciones, concurrencia, programación orientada a objetos y recolección automática de residuos de memoria.

Hemos incluido en este CD-ROM varias implementaciones del lenguaje, algunas de ellas disponibles para más de una plataforma, de tal manera que aquellos interesados, se instalen la más adecuada a sus necesidades.

Smalltalk

Smalltalk es el verdadero lenguaje de programación orientado a objetos, integrado en un rico entorno de desarrollo basado en múltiples ventanas.

Las innovaciones que Smalltalk ofrece, han sido incorporadas a lo largo de los años en la mayoría de los IDEs y en todas las plataformas. Ha sido además, el primero en incorporar un sistema de creación que iba más allá del habitual editar, compilar, montar, y depurar.

Asimismo, hemos incluido un volcado completo del *Smalltalk Archive*, un sitio Web que contiene todas las fuentes necesarias para su instalación sobre cualquier plataforma, así como una completa librería de programas escritos para este lenguaje. Incluimos una interesante variación de este entorno llamada Squeak, disponible para Linux, Macintosh y Windows NT/95.

Instant Basic 4 JavaTM

La creación de aplicaciones en Java ofrece numerosas mejoras, aunque aprender otro lenguaje nuevo o herramienta de desarrollo lleva bastante tiempo, incrementa los costes y puede ralentizar el proceso de desarrollo en casi tres cuartas partes.

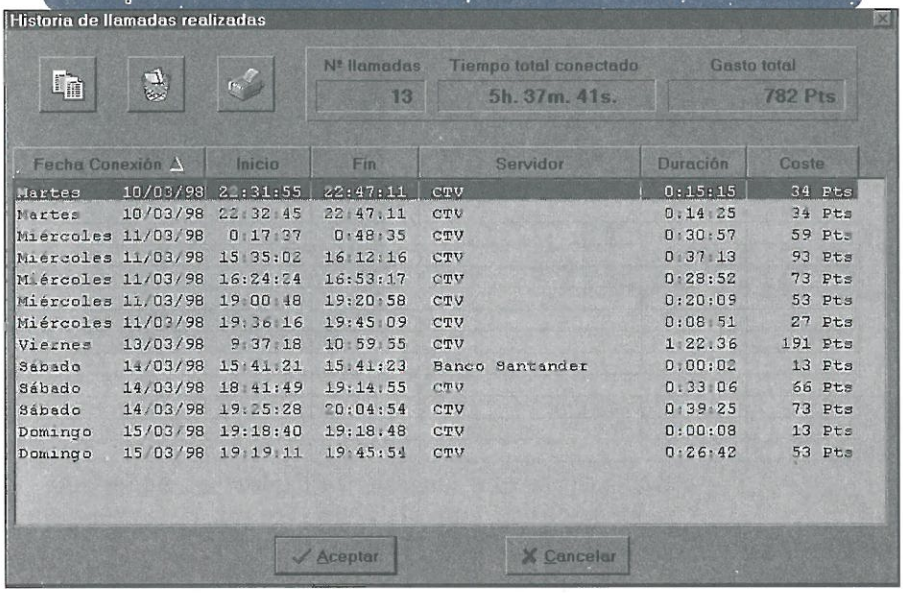
Instant Basic for JavaTM es un IDE (Integrated Development Environment) similar a Microsoft Visual Basic que permite crear fácilmente soluciones 100% Java sin necesidad de aprender Java o un nuevo IDE Java.

Para que podamos instalar esta aplicación, deberemos tener en funcionamiento la versión más reciente del Java Developers Kit de Sun, que actualmente se encuentra en la versión 1.1.5, que también suministramos con esta herramienta.

HTML v4.0

Última revisión de la especificación definitiva de HTML 4.0 recién aprobada por el Word Wide Web Consortium (W3C). Si ésta actual se compara con la versión anterior, la nueva especificación incorpora el ya famoso HTML dinámico, soporte de documentos multilingües, mejoras en formularios, *frames* y tablas, soporte de hojas de estilo. Por otro lado también se puede encontrar

Figura 1.



en ella objetos y *scripts*, así como numerosas ayudas a la navegación para discapacitados.

i.Share v2.5

Todos sus PCs enganchados a la red bajo Windows podrán acceder simultáneamente a Internet desde un único y seguro punto de enlace. i.Share ahorra dinero, tiempo e incrementa la productividad. Es la solución más barata y sencilla que podrá encontrar.

La versión de evaluación incluida se limita a 3 conexiones simultáneas y a 30 días desde la fecha de instalación.

Personal Oracle 8 Multilingual

Oracle 8 es un componente clave en la Network Computing Architecture (NCA), ha sido diseñado como una base de datos distribuida que cumple perfectamente los requerimientos de rendimiento, confianza y escalabilidad establecidas en los métodos orientados a objetos y proceso centralizado de red.

La versión que aquí presentamos es capaz de ejecutarse en Windows 95 y NT. Otra ventaja que ofrece es que dispone de 40 lenguajes y más de 140 conjuntos de caracteres.

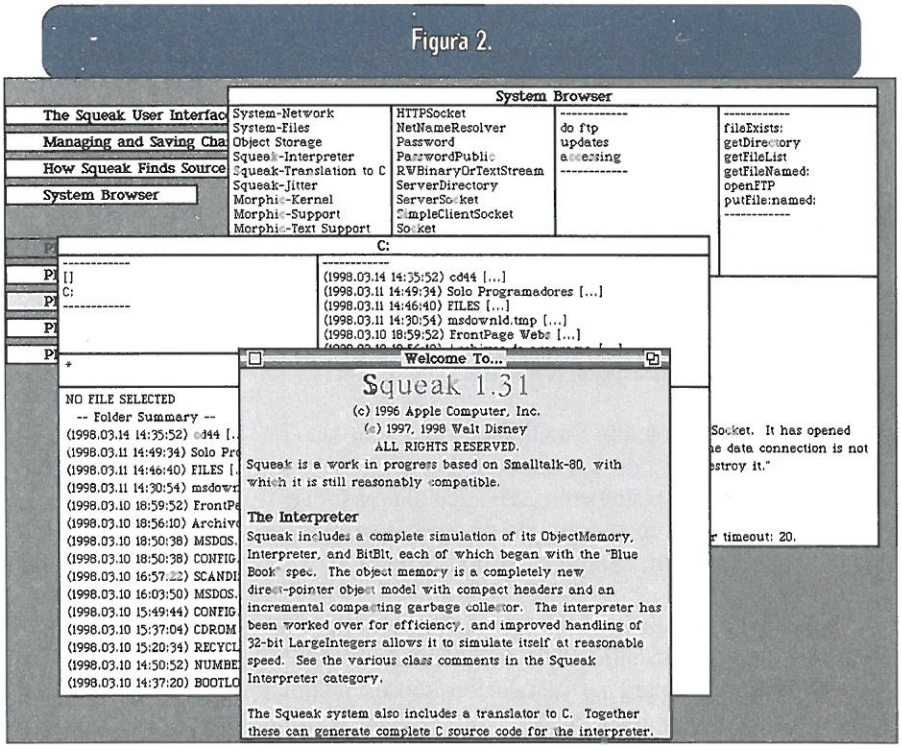
Debido al gran tamaño de la aplicación completa, y para no llenar todo el espacio del CD-ROM de este mes, hemos convertido la distribución de Oracle en un ejecutable que se autodescomprime al disco duro ocupando poco más de 500Mb. Esto no es mucho teniendo en cuenta que habrá de ser instalado en un servidor NT, y ya sabemos en qué tipo de máquinas corre este sistema operativo...

Llamadas32 v4.0

Esta aplicación para Windows 95 se integra perfectamente al entorno, añadiendo un pequeño icono en la barra de tareas. En el momento que se inicie una llamada, se activará y comenzará a contabilizar automáticamente el gasto.

Posteriormente se podrá visualizar una factura detallada con información (duración, coste, ...) de las llamadas realizadas a los diferentes servidores de información utilizando el acceso telefónico a redes de Windows 95.

Figura 2.



PVP 2.995 IVA Incluido

AUTO ESCUELA MULTIMEDIA

NOVEDAD

- **El método más eficaz para alcanzar el éxito seguro.**
- **Aprenda todo lo necesario para poder presentarse con éxito a los exámenes del permiso de conducir.**
- **"Autoescuela Multimedia" le enseña sin esfuerzo todos los conceptos básicos y avanzados que se necesitan para obtener un aprobado.**

A la venta en quioscos y tiendas de informática

C/ Aragoneses, 7. 28.108 Pol. Ind. Alcobendas (MADRID)
Tel.: (91) 661 42 11* Fax: (91) 661 43 86 http://www.towercom.es

ABETO EDITORIAL